www.plcworld.cn

SAMS

附光盘

21天学通C语言（第6版）

〔美〕Bradley L. Jones
Peter Aitken 著
信达工作室 译

人民邮电出版社
POSTS & TELECOMMUNICATIONS PRESS

www.plcworld.cn

美术编辑：胡平利

鸿鹄书院 http://book.hh010.com

TP312C
Q51

21天学C语言（第6版）

让您的编程技能更上一层楼

只需21天，读者便可掌握使用C语言进行编程所需的所有技能。本书是一本完整的C语言教程，它介绍了C语言的基本知识，并阐述了其他高级特性和概念。

- 了解C语言的基本知识；
- 掌握C语言中所有新的和高级特性；
- 通过实用的真实范例，学习如何高效地使用最新的C语言工具和特性；
- 获得权威人士提供的有关在集成环境中实现C语言的专家级技巧。

本书是针对自学而设计的。读者可以按从头到尾的顺序进行阅读，也可以选择阅读最感兴趣的内容。

光盘中包括

- 本书中所有范例的源代码；
- DJGPP和Dev-Shed C++编译器；
- 《Teach Yourself C# in 21 Days》一书（英文版）中几章的内容；
- 本书附加课程的内容。

- 创建自己的程序并使用书中的程序；
- 探索高级C语言编程技术；
- 了解数据类型、循环和字符串，以提高程序的效率；
- 通过学习数组、结构和共用体，提高编程技能；
- 使用日期/时间例程、数学函数和其他标准函数；
- 读写文件（输入和输出）；
- “作业”让读者复习所学的内容；
- “问与答”包含常见的问题及其答案。

附加课程：

- 学习C++、Java和C#的基本概念；
- 了解面向对象语言与C语言之间的差别。

ISBN 7-115-11144-8

9 787115 111449 >

ISBN7-115-11144-8/TP·3357
定价：52.00元（附光盘）

TP3
Q51

人民邮电出版社
http://www.ptpress.com.cn

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

21天学通C语言（第6版）

[美] Bradley L. Jones　Peter Aitken　著
信达工作室　译

人民邮电出版社

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

图书在版编目（CIP）数据

21 天学通 C 语言 /（美）琼斯（Jones, B. L.），（美）艾特肯（Aitken, P.）著；信达工作室译.
—北京：人民邮电出版社，2003.3
书名原文：Teach Yourself C in 21 Days

ISBN 7-115-11144-8

Ⅰ. 2… Ⅱ. ①琼… ②艾… ③信… Ⅲ. C 语言—程序设计
Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2003）第 008590 号

版权声明

Bradley L. Jones, Peter Aitken:Sams Teach Yourself C in 21 Days, Six Edition (ISBN:0672324482)

Copyright © 2003 by Sams Publishing.

Authorized translation from the English language edition published by Sams.

All rights reserved.

本书中文简体字版由美国 **Sams** 授权人民邮电出版社出版，未经出版者书面许可，对本书任何部分不得以任何方式复制或抄袭。

版权所有，侵权必究。

21 天学通 C 语言（第 6 版）

◆ 著 [美] Bradley L. Jones Peter Aitken
译 信达工作室
责任编辑 李 际

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www. ptpress.com.cn
读者热线 010-67132705
北京汉魂图文设计有限公司制作
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：787×1092 1/16
印张：32
字数：1 041 千字 2003 年 3 月第 1 版
印数：1-5 000 册 2003 年 3 月北京第 1 次印刷

著作权合同登记 图字：01 - 2002 - 5929 号

ISBN 7-115-11144-8/TP • 3357

定价：52.00 元（附光盘）

本书如有印装质量问题，请与本社联系 电话：（010）67129223

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

内容提要

本书译自《Teach Yourself C in 21 Days》第6版，该书的前五版都登上了畅销书排行榜，是初学者学习C语言的经典之作。本版按最新的标准（ISO/IEC:9899-1999），以循序渐进的方式介绍了C语言编程方面的知识，并提供了丰富的实例和大量的练习。通过学习实例，并将所学知识用于完成练习，读者将逐步了解、熟悉并精通C语言。

本书包括四周的课程，前三周详细介绍了C语言，第四周是附加课程，简要地介绍了最流行的面向对象语言——C++、Java和C#，附加课程的内容在光盘中以pdf格式文件提供。第一周的课程介绍了C语言程序的基本元素，包括变量、常量、语句、表达式、函数和循环；第二周介绍了数组、指针、字符和字符串、结构和共用体、变量的作用域、输入/输出等；第三周介绍了有关指针和函数的高级主题、磁盘文件读写、字符串操纵函数、函数库、内存管理以及编译器的高级用法等。

本书是为初中级程序员编写的，可作为学习C语言的教程或参考资料。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

作者简介

Bradley L. Jones 在 internet.com 网站就职，负责管理 EarthWeb 软件开发频道，包括诸如 Developer.com、CodeGuru.com 和 Gamelan.com 等网站。Bradley 领导开发了用于各种平台（从 Palm OS 到大型机系统）的小型和分布式系统。Bradley 拥有使用 C、C#、C++、XML、SQL Server、PowerBuilder、Visual Basic、ASP 和 Satellite Forms 等工具进行开发工作的经验。他编写的其他图书包括《Teach Yourself Advanced C in 21 Days》和《21 天学通 C#》。

Peter Aitken 拥有十多年计算机和编程方面的写作经验，编写的图书大约有 30 本，并在计算机杂志上发表过几百篇文章。他最近编写的图书有《Visual Basic .NET Programming With Peter Aitken》、《Office XP Development With VBA》、《XML the Microsoft Way》、《Windows Script Host》和《Teach Yourself Visual Basic .NET Internet Programming in 21 Days》等。Peter Aitken 曾担任《Visual Developer》杂志的撰稿编辑多年，负责编写 Visual Basic 专栏；另外，他还经常在《Microsoft OfficePro》杂志和网站 DevX 上发表文章。从 1994 年起，Peter Aitken 一直负责经营 PGA 咨询公司，为企业、学术和政府机构开发应用程序和 Internet 程序。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

前　言

本书旨在引导读者在21天内学通C语言编程。虽然有来自诸如C++、Java和C#的激烈竞争，但很多初学编程者还是会选择C语言。基于第1天课程介绍的原因，选择C语言可确保您不会误入“歧途”。

将本书作为自学C语言的教材是一个明智的决定。虽然市面上有很多有关C语言的图书，但本书介绍C语言的方式最为合理，也让读者学习起来最为容易。本书的前五版都登上了畅销书排行榜，这一事实表明我们的观点得到了读者的认同。本书是按读者每天阅读一章的方式编写的。读者不需要有任何编程经验，但如果读者以前学习过其他语言（如BASIC），学习起来将更快。另外，本书介绍的是C语言，而不针对任何编译器和计算机——读者使用的是PC、Mac还是UNIX系统将无关紧要。

本书包含一周的附加课程，旨在让读者对面向对象编程以及最流行的面向对象语言（C++、Java和C#）有个初步的了解。虽然这些章节无法全面地介绍这些主题，但将让您得以起步。

本书的特色

本书包含诸如语法、提示、注释、警告等内容，它们将给读者以启迪。“语法”介绍如何使用特定的C语言概念，对C语言命令和概念做了全面的阐述，并提供了范例。下面是一个“语法”范例：

printf()函数的语法如下：

```
#include <stdio.h>
printf(format-string[,arguments,…]);
```

printf()是一个函数，它接受一系列的参数(arguments)，其中每个参数用于给定格式化字符串中的一个转换说明符。该函数将格式化信息显示到标准输出设备（通常为屏幕）中。要使用printf()，必须包含标准输入/输出头文件stdio.h。

format-string是必不可少的；但arguments是可选的。对于每一个参数，格式化字符串中都必须有一个转换说明符与之对应。格式化字符串可以包含转义序列。下面是几个调用printf()的范例及其输出：

范例1：

```
#include <stdio.h>
int main( void )
{
   printf("This is an example of something printed! ");
}
```

输出：

```
This is an example of something printed!
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

范例 2：

```
printf("This prints a character, %c\na number, %d\na floating point,
%f",'z', 123, 456.789 );
```

输出：

```
This prints a character, z
a number, 123
a floating point, 456.789
```

本书的另一个特色是“应该/不应该”，它告诉您应该做什么，不应该做什么。

应　该	不 应 该
应阅读本节余下的内容，它解释了每个课程最后的作业。	不要略过任何小测验和练习。如果能够完成当天的作业，则说明您已经为学习新的内容做好了准备。

本书中还有“注意”、“提示”和“警告”。“提示”提供了使用 C 语言的技巧和捷径；“注意”说明了一些特殊的细节，对 C 语言概念做了进一步的解释；“警告”帮助您避免发生一些潜在的问题。

本书通过大量的范例程序来说明 C 语言的特性和概念，让您能够在自己的程序中使用它们。对每个范例程序的讨论都由三部分组成：程序代码、用户输入和程序输出、对程序工作原理的分析。这些部分都由特殊的图标标识。

每章的最后都有“问与答”，其中包含与本章内容相关的常见问题及其答案。另外，每章的最后还有作业，其中包含小测验和练习。小测验检查读者对本章介绍的概念的理解程度。如果要检查自己的答案是否正确，或者对问题回答不上来，可参考附录 F 的答案。

仅仅阅读本书并不能学会 C 语言。要成为程序员，您必须编写程序。为此，我们在每一个小测验的后面提供了一组练习，建议您完成每一个练习。学习 C 语言的最佳方式是编写 C 语言代码。

我们认为，排错练习最有帮助。bug 是 C 中的一种程序错误，排错练习提供了一些代码，其中包含常见的问题（错误），您必须找出并更正这些错误。如果有困难，可参阅附录 F 提供的答案。

随着对 C 语言的介绍越来越深入，有些练习的答案将很长，而有些练习有多种解决方案。因此，对于最后几章中的有些练习，附录 F 没有提供答案。

本版所做的改进

任何事情都不是十全十美的，但我们尽可能追求完美。本书是《21 天学通 C 语言》的第 6 版。编写本书时，我们做了更大的努力，使其中的代码与更多的 C 语言编译器兼容。我们对本书做了多次审校，确保它在技术上极其准确。另外，根据读者的指正，我们更正了前几版中的很多错误。



注意：我们在以下平台上对本书中的源代码进行了编译和测试：DOS、Windows、Macintosh、UNIX、Linux 和 OS/2。另外，本书以前版本的读者已经将这些代码用于了几乎所有支持 C 语言的平台中。

本版的另一项特色是 Type & Run 内容。本书共有 6 个 Type & Run，其中每个都包含一个简短的 C 语言程序。这些程序演示了 C 语言编程技术，同时能够完成一些有趣且有用的工作。您可以输入并运行这些程序。输入这些程序后，可以对其中的代码进行修改，看看能够使之完成其他哪些工作。Type & Run 就是提供给您进行试验的，希望您试验愉快！

附带光盘

为方便读者，附带光盘中包含了本书中所有的代码；另外，还有几个编译器，其中包括 Dev-C++——附

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

求G介绍了如何使用该编译器。

源代码的勘误和更新将在网站 www.samspblishing.com 上发布（如果有的话）。

本书的约定

本书使用不同的字体来区分C语言代码和正文以及指出一些重要的概念。源代码的字体为 Courier New；在程序的输出中，粗体表示用户输入的内容；占位符（应替换为实际的代码）则为斜体。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

目　录

第一周课程

第 1 天课程　C 语言初步 ……2

1.1　C 语言简史 ……2
1.2　为何要使用 C 语言 ……2
1.3　编程前的准备工作 ……3
1.4　程序开发周期 ……4
1.4.1　创建源代码 ……4
1.4.2　编译源代码 ……4
1.4.3　链接以创建可执行文件 ……5
1.4.4　结束开发周期 ……6
1.5　第一个 C 语言程序 ……7
1.5.1　输入并编译 hello.c ……7
1.6　总结 ……9
1.7　问与答 ……9
1.8　作业 …… 10
1.8.1　小测验 …… 10
1.8.2　练习 …… 10

TYPE&RUN1　打印程序清单 …… 12

第一个 TYPE&RUN …… 12

第 2 天课程　C 语言程序的组成部分 …… 14

2.1　一个简短的 C 语言程序 …… 14
2.2　程序的组成部分 …… 15
2.2.1　main()函数（第 8～23 行） …… 15
2.2.2　#include 编译指令（第 2 行） …… 15
2.2.3　变量定义（第 4 行） …… 15
2.2.4　函数原型（第 6 行） …… 16
2.2.5　程序语句（第 11、12、15、16、19、20、22 和 28 行） … 16
2.2.6　函数定义（第 26～29 行） …… 16
2.2.7　程序注释（第 1、10、14、18 和 25 行） …… 16
2.2.8　使用花括号（第 9、23、27 和 29 行） …… 17
2.2.9　运行程序 …… 17

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

2.2.10 有关精度的说明 …… 18
2.3 重温程序的组成部分 …… 18
2.4 总结 …… 20
2.5 问与答 …… 20
2.6 作业 …… 20
2.6.1 小测验 …… 20
2.6.2 练习 …… 21

第 3 天课程 存储信息：变量和常量 …… 23

3.1 计算机内存 …… 23
3.2 使用变量存储信息 …… 24
3.2.1 变量名 …… 24
3.3 数值变量的类型 …… 25
3.3.1 变量声明 …… 27
3.3.2 typedef 关键字 …… 27
3.3.3 初始化变量 …… 28
3.4 常量 …… 28
3.4.1 字面常量 …… 28
3.4.2 符号常量 …… 29
3.5 总结 …… 31
3.6 问与答 …… 32
3.7 作业 …… 32
3.7.1 小测验 …… 32
3.7.2 练习 …… 32

第 4 天课程 语句、表达式和运算符 …… 34

4.1 语句 …… 34
4.1.1 空白对语句的影响 …… 34
4.1.2 创建空语句 …… 35
4.1.3 使用复合语句 …… 35
4.2 表达式 …… 35
4.2.1 简单表达式 …… 35
4.2.2 复杂表达式 …… 36
4.3 运算符 …… 36
4.3.1 赋值运算符 …… 36
4.3.2 数学运算符 …… 37
4.3.3 运算符优先级和圆括号 …… 40
4.3.4 子表达式的计算顺序 …… 41
4.3.5 关系运算符 …… 41
4.4 if 语句 …… 42
4.4.1 else 子句 …… 44
4.5 判断关系表达式 …… 46
4.5.1 关系运算符的优先级 …… 47
4.6 逻辑运算符 …… 48

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

4.7　再谈 true/false 值 …… 48
4.7.1　运算符的优先级 …… 49
4.7.2　复合赋值运算符 …… 50
4.7.3　条件运算符 …… 50
4.7.4　逗号运算符 …… 51
4.8　再谈运算符优先级 …… 51
4.9　总结 …… 52
4.10　问与答 …… 52
4.11　作业 …… 53
4.11.1　小测验 …… 53
4.11.2　练习 …… 53

TYPE&RUN2　猜数游戏 …… 55

第 5 天课程　使用函数封装代码 …… 57

5.1　函数是什么 …… 57
5.1.1　函数的定义 …… 57
5.1.2　函数的用法 …… 57
5.2　函数的工作原理 …… 59
5.3　函数和结构化编程 …… 60
5.3.1　结构化编程的优点 …… 60
5.3.2　规划结构化程序 …… 60
5.3.3　从顶向下的方法 …… 61
5.4　编写函数 …… 61
5.4.1　函数头 …… 62
5.4.2　函数的返回类型 …… 62
5.4.3　函数名 …… 62
5.4.4　参数列表 …… 62
5.4.5　函数体 …… 64
5.4.6　函数原型 …… 67
5.5　将参数传递给函数 …… 67
5.6　调用函数 …… 68
5.6.1　递归 …… 69
5.7　函数的位置 …… 70
5.8　内联函数 …… 70
5.9　总结 …… 71
5.10　问与答 …… 71
5.11　作业 …… 72
5.11.1　小测验 …… 72
5.11.2　练习 …… 72

第 6 天课程　基本的程序流程控制 …… 74

6.1　数组的基本知识 …… 74
6.2　控制程序的执行 …… 75

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

6.2.1 for 语句 …… 75
6.2.2 嵌套 for 语句 …… 79
6.2.3 while 语句 …… 80
6.2.4 嵌套 while 语句 …… 82
6.2.5 do...while 循环 …… 84
6.3 嵌套循环 …… 87
6.4 总结 …… 88
6.5 问与答 …… 88
6.6 作业 …… 88
6.6.1 小测验 …… 89
6.6.2 练习 …… 89

第 7 天课程 信息读写基础 …… 90

7.1 在屏幕上显示信息 …… 90
7.1.1 printf()函数 …… 90
7.1.2 格式化字符串 …… 90
7.1.3 转义序列 …… 91
7.1.4 使用 puts()显示消息 …… 96
7.2 使用 scanf()函数输入数值数据 …… 97
7.3 三字符序列 …… 100
7.4 总结 …… 101
7.5 问与答 …… 101
7.6 作业 …… 102
7.6.1 小测验 …… 102
7.6.2 练习 …… 102

第一周复习 …… 104

第二周课程

第 8 天课程 使用数值数组 …… 110

8.1 数组是什么 …… 110
8.1.1 一维数组 …… 110
8.1.2 多维数组 …… 113
8.2 命名和声明数组 …… 114
8.2.1 初始化数组 …… 115
8.2.2 初始化多维数组 …… 116
8.2.3 数组的最大长度 …… 119
8.3 总结 …… 121
8.4 问与答 …… 121
8.5 作业 …… 121
8.5.1 小测验 …… 121

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

8.5.2 练习 …… 122

第9天课程　指针 …… 123

9.1 指针是什么 …… 123
9.1.1 计算机内存 …… 123
9.1.2 创建指针 …… 123
9.2 指针和简单变量 …… 124
9.2.1 声明指针 …… 124
9.2.2 初始化指针 …… 124
9.2.3 使用指针 …… 125
9.3 指针和变量类型 …… 126
9.4 指针和数组 …… 127
9.4.1 作为指针的数组名 …… 128
9.4.2 数组元素的存储 …… 128
9.4.3 指针算术 …… 130
9.5 有关指针的注意事项 …… 133
9.6 数组下标表示法和指针 …… 133
9.7 将数组传递给函数 …… 133
9.8 总结 …… 137
9.9 问与答 …… 137
9.10 作业 …… 138
9.10.1 小测验 …… 138
9.10.2 练习 …… 138

TYPE&RUN3 让程序暂停 …… 139

第10天课程　字符和字符串 …… 141

10.1 char数据类型 …… 141
10.2 使用字符变量 …… 141
10.3 使用字符串 …… 144
10.3.1 字符数组 …… 144
10.3.2 初始化字符数组 …… 144
10.4 字符串和指针 …… 144
10.5 不存储在数组中的字符串 …… 145
10.5.1 编译时分配字符空间 …… 145
10.5.2 malloc()函数 …… 145
10.5.3 使用 malloc()函数 …… 146
10.6 显示字符串和字符 …… 148
10.6.1 puts()函数 …… 149
10.6.2 printf()函数 …… 149
10.7 从键盘读取字符串 …… 150
10.7.1 使用 gets()函数输入字符串 …… 150
10.7.2 使用 scanf()函数输入字符串 …… 152
10.8 总结 …… 154

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

10.9 问与答 …… 154
10.10 作业 …… 155
10.10.1 小测验 …… 155
10.10.2 练习 …… 156

第 11 天课程 结构、共用体和 TypeDef …… 157

11.1 简单结构 …… 157
11.1.1 定义和声明结构 …… 157
11.1.2 存取结构的成员 …… 158
11.2 复杂结构 …… 160
11.2.1 包含其他结构的结构 …… 160
11.2.2 包含数组的结构 …… 162
11.3 结构数组 …… 164
11.4 初始化结构 …… 167
11.5 结构和指针 …… 168
11.5.1 将指针作为结构的成员 …… 168
11.5.2 创建指向结构的指针 …… 170
11.5.3 使用指针和结构数组 …… 171
11.5.4 将结构作为参数传递给函数 …… 173
11.6 共用体 …… 174
11.6.1 定义、声明和初始化共用体 …… 175
11.6.2 存取共用体的成员 …… 175
11.7 使用 typedef 给结构创建别名 …… 179
11.8 总结 …… 179
11.9 问与答 …… 179
11.10 作业 …… 180
11.10.1 小测验 …… 180
11.10.2 练习 …… 180

第 12 天课程 变量作用域 …… 182

12.1 作用域是什么 …… 182
12.1.1 演示作用域 …… 182
12.1.2 作用域为何重要 …… 184
12.2 外部变量 …… 184
12.2.1 外部变量的作用域 …… 184
12.2.2 何时使用外部变量 …… 184
12.2.3 extern 关键字 …… 184
12.3 局部变量 …… 185
12.3.1 静态变量和动态变量 …… 186
12.3.2 函数参数的作用域 …… 187
12.3.3 外部静态变量 …… 188
12.3.4 寄存器变量 …… 188
12.4 局部变量和 main()函数 …… 189
12.5 应使用哪种存储类型 …… 189

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

12.6　局部变量和代码块 …… 189
12.7　总结 …… 190
12.8　问与答 …… 191
12.9　作业 …… 191
12.9.1　小测验 …… 191
12.9.2　练习 …… 191

TYPE&RUN4　机密消息 …… 194

第 13 天课程　高级程序流程控制 …… 197

13.1　提早结束循环 …… 197
13.1.1　break 语句 …… 197
13.1.2　continue 语句 …… 199
13.2　goto 语句 …… 200
13.3　死循环 …… 202
13.4　switch 语句 …… 205
13.5　退出程序 …… 212
13.5.1　exit()函数 …… 212
13.6　在程序中执行操作系统命令 …… 213
13.7　总结 …… 214
13.8　问与答 …… 215
13.9　作业 …… 215
13.9.1　小测验 …… 215
13.9.2　练习 …… 215

第 14 天课程　操纵屏幕、打印机和键盘 …… 217

14.1　流和 C 语言 …… 217
14.1.1　何为程序的输入/输出 …… 217
14.1.2　什么是流 …… 217
14.1.3　文本流和二进制流 …… 218
14.1.4　预定义的流 …… 218
14.2　使用 C 语言的流函数 …… 219
14.2.1　例子 …… 219
14.3　读取键盘输入 …… 220
14.3.1　字符输入 …… 220
14.3.2　格式化输入 …… 225
14.4　控制屏幕输出 …… 231
14.4.1　使用 putchar()、putc()和 fputc()输出字符 …… 231
14.4.2　使用 puts()和 fputs()输出字符串 …… 233
14.4.3　使用 printf()和 fprintf()格式化输出 …… 234
14.5　重定向输入/输出 …… 238
14.5.1　重定向输入 …… 239
14.6　何时使用 fprintf() …… 239
14.6.1　使用 stderr …… 240

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

14.7 总结 …… 241
14.8 问与答 …… 241
14.9 作业 …… 242
14.9.1 小测验 …… 242
14.9.2 练习 …… 242

第二周复习 …… 244

第三周课程

第 15 天课程 有关指针的高级主题 …… 252

15.1 声明指向指针的指针 …… 252
15.2 指针和多维数组 …… 253
15.3 指针数组 …… 259
15.3.1 复习字符串和指针 …… 259
15.3.2 声明 char 类型指针数组 …… 259
15.3.3 范例 …… 261
15.4 函数指针 …… 265
15.4.1 声明函数指针 …… 265
15.4.2 初始化并使用函数指针 …… 266
15.5 链表 …… 273
15.5.1 有关链表的基本知识 …… 273
15.5.2 使用链表 …… 274
15.5.3 演示简单链表 …… 278
15.5.4 实现链表 …… 280
15.6 总结 …… 286
15.7 问与答 …… 286
15.8 作业 …… 286
15.8.1 小测验 …… 287
15.8.2 练习 …… 287

第 16 天课程 使用磁盘文件 …… 289

16.1 将流与磁盘文件关联起来 …… 289
16.2 磁盘文件的类型 …… 289
16.3 文件名 …… 290
16.4 打开文件 …… 290
16.5 读写文件数据 …… 292
16.5.1 格式化文件输入/输出 …… 293
16.5.2 字符输入/输出 …… 296
16.5.3 直接文件输入/输出 …… 297
16.6 文件缓冲技术：关闭和刷新文件 …… 300
16.7 顺序文件存取和随机文件存取 …… 301

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

16.7.1 ftell()和 rewind()函数 …… 301
16.7.2 fseek()函数 …… 303
16.8 检测文件尾 …… 305
16.9 文件管理函数 …… 307
16.9.1 删除文件 …… 307
16.9.2 给文件重命名 …… 308
16.9.3 复制文件 …… 309
16.10 使用临时文件 …… 311
16.11 总结 …… 312
16.12 问与答 …… 312
16.13 作业 …… 313
16.13.1 小测验 …… 313
16.13.2 练习 …… 313
TYPE&RUN5 计算字符数 …… 314

第 17 天课程 操纵字符串 …… 318

17.1 确定字符串的长度 …… 318
17.2 复制字符串 …… 319
17.2.1 strcpy()函数 …… 319
17.2.2 strncpy()函数 …… 320
17.2.3 strdup()函数 …… 321
17.3 拼接字符串 …… 322
17.3.1 strcat()函数 …… 322
17.3.2 strncat()函数 …… 324
17.4 比较字符串 …… 325
17.4.1 比较两个完整字符串 …… 325
17.4.2 比较字符串的一部分 …… 326
17.4.3 比较字符串时忽略大小写 …… 328
17.5 查找字符串 …… 328
17.5.1 strchr()函数 …… 328
17.5.2 strrchr()函数 …… 329
17.5.3 strcspn()函数 …… 329
17.5.4 strspn()函数 …… 330
17.5.5 strpbrk()函数 …… 331
17.5.6 strstr()函数 …… 331
17.6 字符串转换 …… 332
17.7 其他字符串函数 …… 333
17.7.1 strrev()函数 …… 333
17.7.2 strset()和 strnset()函数 …… 334
17.8 将字符串转换为数字 …… 334
17.8.1 将字符串转换为 int …… 335
17.8.2 将字符串转换为 long 值 …… 335
17.8.3 将字符串转换为 long long 值 …… 335

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

17.8.4 将字符串转换为浮点数 …… 335
17.9 字符检测函数 …… 336
17.9.1 ANSI 对大小写转换的支持 …… 339
17.10 总结 …… 340
17.11 问与答 …… 340
17.12 作业 …… 341
17.12.1 小测验 …… 341
17.12.2 练习 …… 341

第 18 天课程 有关函数的高级主题 …… 343

18.1 将指针传递给函数 …… 343
18.2 void 类型的指针 …… 346
18.3 接受可变数目参数的函数 …… 348
18.4 返回指针的函数 …… 350
18.5 总结 …… 352
18.6 问与答 …… 352
18.7 作业 …… 352
18.7.1 小测验 …… 352
18.7.2 练习 …… 353

第 19 天课程 函数库 …… 354

19.1 数学函数 …… 354
19.1.1 三角函数 …… 354
19.1.2 指数函数和对数函数 …… 354
19.1.3 双曲线函数 …… 355
19.1.4 其他数学函数 …… 355
19.1.5 演示数学函数 …… 355
19.2 处理时间 …… 356
19.2.1 时间的表示 …… 356
19.2.2 时间函数 …… 356
19.2.3 使用时间函数 …… 359
19.3 处理错误 …… 360
19.3.1 assert()宏 …… 361
19.3.2 头文件 errno.h …… 362
19.3.3 perror()函数 …… 362
19.4 查找和排序 …… 364
19.4.1 使用 bsearch()进行查找 …… 364
19.4.2 使用 qsort()进行排序 …… 365
19.4.3 演示查找和排序 …… 365
19.5 总结 …… 370
19.6 问与答 …… 370
19.7 作业 …… 370
19.7.1 小测验 …… 370
19.7.2 练习 …… 371

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

TYPE&RUN6　计算抵押贷款的偿还金额 …… 372

第 20 天课程　管理内存 …… 374

20.1　类型转换 …… 374
20.1.1　自动类型转换 …… 374
20.1.2　显式转换 …… 376
20.2　分配内存的存储空间 …… 377
20.2.1　使用 malloc()函数分配内存 …… 377
20.2.2　使用 calloc()函数分配内存 …… 379
20.2.3　使用 realloc()函数分配更多的内存 …… 380
20.2.4　使用 free()函数释放内存 …… 381
20.3　操纵内存块 …… 383
20.3.1　使用 memset()函数初始化内存 …… 383
20.3.2　使用 memcpy()复制内存中的数据 …… 383
20.3.3　使用 memmove()函数移动内存中的数据 …… 383
20.4　位的用法 …… 385
20.4.1　移位运算符 …… 385
20.4.2　按位逻辑运算符 …… 386
20.4.3　求补运算符 …… 387
20.4.4　结构中的位字段 …… 387
20.5　总结 …… 389
20.6　问与答 …… 389
20.7　作业 …… 390
20.7.1　小测验 …… 390
20.7.2　练习 …… 390

第 21 天课程　编译器的高级用法 …… 392

21.1　使用多个源代码文件的编程 …… 392
21.1.1　模块化编程的优点 …… 392
21.1.2　模块化编程技术 …… 392
21.1.3　模块的组成部分 …… 394
21.1.4　外部变量和模块化编程 …… 395
21.1.5　使用.obj 文件 …… 395
21.1.6　使用生成工具 …… 396
21.2　C 语言的预处理器 …… 396
21.2.1　#define 预处理器编译指令 …… 397
21.2.2　使用编译指令#include …… 400
21.2.3　使用#if、#elif、#else 和#endif …… 400
21.2.4　使用#if...#endif 来帮助调试 …… 401
21.2.5　避免将头文件包含多次 …… 401
21.2.6　#undef 编译指令 …… 402
21.3　预定义的宏 …… 402
21.4　使用命令行参数 …… 403
21.5　总结 …… 405

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

21.6 问与答 …… 405
21.7 作业 …… 405
21.7.1 小测验 …… 405
21.7.2 练习 …… 406

第三周复习 …… 407

附加课程（具体内容见光盘） …… 413

附录 A ASCII 字符集 …… 414

附录 B C/C++中的保留字 …… 418

附录 C 使用二进制和十六进制数 …… 420

C.1 十进制 …… 420
C.2 二进制 …… 420
C.3 十六进制 …… 420

附录 D 移植性问题 …… 422

D.1 ANSI 标准 …… 422
D.2 ANSI 关键字 …… 422
D.3 区分大小写 …… 422
D.4 可移植的字符 …… 424
D.5 确保 ANSI 兼容性 …… 424
D.6 绕开 ANSI 标准 …… 424
D.7 使用可移植的数值变量 …… 425
D.7.1 最大值和最小值 …… 426
D.7.2 确定数字的类型 …… 430
D.7.3 转换字符的大小写：一个可移植性范例 …… 434
D.8 可移植的结构和共用体 …… 434
D.8.1 字对齐 …… 434
D.8.2 读写结构 …… 435
D.8.3 在可移植的程序中使用非-ANSI 特性 …… 436
D.8.4 ANSI 标准头文件 …… 437
D.9 总结 …… 438
D.10 问与答 …… 438
D.11 作业 …… 438
D.11.1 小测验 …… 438
D.11.2 练习 …… 439

附录 E 常用的 C 语言函数 …… 440

附录 F 作业答案 …… 444

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 1 天课程的答案 …… 444
小测验 …… 444
练习 …… 444
第 2 天课程的答案 …… 445
小测验 …… 445
练习 …… 445
第 3 天课程的答案 …… 446
小测验 …… 446
练习 …… 446
第 4 天课程的答案 …… 447
小测验 …… 447
练习 …… 447
第 5 天课程的答案 …… 449
小测验 …… 449
练习 …… 449
第 6 天课程的答案 …… 452
小测验 …… 452
练习 …… 452
第 7 天课程的答案 …… 453
小测验 …… 453
练习 …… 453
第 8 天课程的答案 …… 457
小测验 …… 457
练习 …… 457
第 9 天课程的答案 …… 461
小测验 …… 461
练习 …… 461
第 10 天课程的答案 …… 463
小测验 …… 463
练习 …… 463
第 11 天课程的答案 …… 465
小测验 …… 465
练习 …… 466
第 12 天课程的答案 …… 467
小测验 …… 467
练习 …… 467
第 13 天课程的答案 …… 470
小测验 …… 470
练习 …… 471
第 14 天课程的答案 …… 471
小测验 …… 471
练习 …… 472
第 15 天课程的答案 …… 472
小测验 …… 472
练习 …… 473

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 16 天课程的答案 …… 473
小测验 …… 473
练习 …… 474
第 17 天课程的答案 …… 474
小测验 …… 474
练习 …… 474
第 18 天课程的答案 …… 475
小测验 …… 475
练习 …… 475
第 19 天课程的答案 …… 475
小测验 …… 475
练习 …… 476
第 20 天课程的答案 …… 476
小测验 …… 476
练习 …… 477
第 21 天课程的答案 …… 477
小测验 …… 477
附加课程 1 的答案 …… 478
小测验 …… 478
附加课程 2 的答案 …… 478
小测验 …… 478
附加课程 3 的答案 …… 478
小测验 …… 478
附加课程 4 的答案 …… 479
小测验 …… 479
附加课程 5 的答案 …… 479
小测验 …… 479
附加课程 6 的答案 …… 480
小测验 …… 480
附加课程 7 的答案 …… 480
小测验 …… 480
练习 …… 480

附录 G Dev-C++编译器 …… 482

G.1 Dev-C++简介 …… 482
G.2 在 Microsoft Windows 上安装 Dev-C++ …… 482
G.3 Dev-C++中的程序 …… 484
G.4 使用 Dev-C++ …… 484
G.4.1 针对 C 语言编程定制 Dev-C++ …… 485
G.4.2 在 Dev-C++中输入并编译程序 …… 486
G.4.3 编译 Dev-C++程序 …… 488
G.4.4 运行 Dev-C++程序 …… 488
G.5 总结 …… 489

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第一周课程

在学习如何使用C语言进行编程之前，读者需要有几样东西：编译器、编辑器以及本书。本书附带的光盘中包含两个编辑器——Bloodshed Dev-C++和DJGPPP，它们都提供了编译器。附录G“Bloodshed Dev-C++编译器”介绍了如何安装Windows环境下的Bloodshed Dev-C++编译器。光盘中的readme文件提供了有关DJGPP的信息。如果读者已经有编辑器和编译器，则无须使用附带光盘中的工具。实际上，由于本书是基于ANSI标准编写的，因此读者可以使用任何标准的C语言编译器和编辑器。

学习计算机语言的最佳方式是，不但要阅读有关这种语言的图书，还应输入并运行大量的程序。本书包含的很多C语言程序让读者能够进行实际练习。如果读者想省事，可以在本书附带的光盘中找到大部分完整的程序清单。虽然读者可以从光盘中复制并运行这些程序，但作者建议您在编辑器中输入这些程序。这将需要更多的时间，但深入地查看这些程序有助于读者理解其中的代码。

本周的课程将介绍完全理解C语言所需的基本知识。在第1天的课程和第2天的课程中，读者将学习如何创建C语言程序，并了解简单C语言程序的组成部分。第3天的课程以前两天介绍的知识为基础，阐述了变量的类型。第4天的课程通过变量和简单的表达式来创建新的值，同时介绍了如何通过if语句做出决策并改变程序的流程。第5天的课程介绍了C语言函数和结构化编程技术。第6天的课程介绍了其他能够用于控制程序流程的命令。本周最后一天的课程讨论了如何打印信息以及如何让程序同键盘和屏幕交互。

除了上述内容外，还包含几个Type & Run，它们位于第1天和第2天以及第4天和第5天的课程之间。第1个Type & Run中的程序用于按行编号打印程序清单，其中包含行号。第2个Type & Run中包含的程序清单用于玩一个“猜数”游戏。

另外，每天课程的最后都提供了作业，其中包含小测验和练习。阅读完每天的课程后，读者应该能够回答小测验中的问题，并完成练习。附录F提供了前20天课程中小测验和练习的答案；而对于第21天课程及附加课程，由于其中的练习的解决方案很多，因此只提供了小测验的答案。强烈建议读者完成这些练习，并检查自己的答案。

虽然本周包含的内容很多，但每次完成一天的课程应该没有问题。

注意：本书是按照ISO/IEC 9899:1999定义的ANSI标准C编写的，因此读者可以使用任何遵守该ANSI标准的C语言编译器。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 1 天课程　C 语言初步

欢迎阅读《21 天学通 C 语言》第 6 版。今天的课程将带领读者开始成为精通 C 语言的程序员之旅。今天您将学习以下内容：

- 为什么说 C 语言是一种不错的编程语言。
- 程序开发周期中的各个步骤。
- 编写、编译并运行您的第一个 C 语言程序。
- 由编译器和链接程序所生成的错误消息。

1.1　C 语言简史

读者可能想知道 C 语言的起源，为何叫 C 语言。C 语言是由贝尔实验室的 Dennis Ritchie 于 1972 年开发的。开发它并非为了消遣，而是有特定的目的：设计 UNIX 操作系统（很多计算机都使用该操作系统）。从一开始，C 语言就是为帮助繁忙的程序员完成其工作而开发的。

由于 C 语言功能强大而灵活，因此很快被传播到贝尔实验室之外，世界各地的程序员都使用它来编写各种程序。然而，不久后，不同的组织便开始使用自己的 C 语言版本，不同实现之间微妙的差别令程序员头痛。为解决这种问题，美国国家标准化组织（ANSI）于 1983 年成立了一个委员会，以确定 C 语言的标准定义——ANSI 标准 C 语言。现代的 C 语言编译器绝大多数都遵守该标准。



注意：虽然 C 语言的变化很小，最近的修改是在 1999 年通过标准 ANSI C-99 进行的。该标准新增了一些特性，本书将对其进行介绍。然而您将发现，老式编译器不支持这些最新的标准。

那么，C 语言的名称是如何来的呢？之所以被称为 C 语言，是因为其前身为 B 语言。B 语言是由贝尔实验室的 Ken Thompson 开发的。您应该猜得到，它为何叫 B 语言。

1.2　为何要使用 C 语言

在当前的计算机编程领域中，有大量的高级语言可供选择，如 C、Perl、BASIC、Java 和 C#。这些都是非常卓越的语言，适合用于完成大部分编程任务。虽然如此，但基于以下几个原因，很多计算机专业人员认为 C 语言是其中最佳的：

- C 语言功能强大、灵活。使用 C 语言能够完成的工作只受限于您的想象力，语言本身不会给您带来任何约束。C 语言可用于完成操作系统、字处理器、图形、电子表格等项目，甚至可用于编写其他语言的编译器。
- C 语言很流行，是专业程序员的首选。因此市面上有大量的 C 语言编译器和附件可供选择。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

- C 语言是可移植的。这意味着为一种计算机系统（如 IBM PC）编写的 C 语言程序，可以在其他系统（极有可能是 DEC VAX 系统）中编译并运行，而只须做少量的修改，甚至无须修改。另外，在使用 Microsoft Windows 操作系统的机器上编写的程序，可以被移植到运行 Linux 的机器中，而只须做少量的修改，甚至无须修改。C 语言的 ANSI 标准（有关编译器的一组规则）进一步改善了可移植性。
- C 语言中的单词很少，包含的术语（称为关键字，C 语言以此为基础来构建其功能）很少。您可能认为，语言包含的关键字（有时候被称为保留字）越多，其功能将越强大。情况并非如此。当您使用 C 语言进行编程时将发现，它能够完成任何任务。
- C 语言是模块化的。可以以例程（被称为函数）的方式来编写 C 语言代码，并在其他应用或程序中再次使用这些函数。通过将一些信息传递给函数，可以创建很有用的、可重用代码。

由于其上述特性，C 语言是编程语言初学者的首选。那么，C++呢？读者可能听说过 C++和面向对象编程。因此，可能会问，C 和 C++之间有何区别，是否应自学 C++而不是 C 语言？

C++是 C 语言的超集，包含了 C 语言中的所有内容，同时增加了面向对象编程方面的内容。当您学习 C++时，几乎有关 C 语言的所有知识都适用。学习 C 语言时，您不但是在学习当今最强大、最流行的编程语言，同时还为面向对象编程做准备。

另一种备受人们关注的语言是 Java。和 C++一样，Java 也是基于 C 语言的。如果读者以后决定学习 Java，将发现几乎有关 C 语言的所有知识都适用。

在这些语言中，最新的是 C#（读作“see-sharp”）。同 C++和 Java 一样，C#也是从 C 语言派生而来的一种面向对象语言。同样，您将发现，很多有关 C 语言的知识也适用于 C#编程。



> 注意：很多人学习 C 语言后，将选择学习 C++、Java 或 C#。因此，在这一版中增加了几天的附加课程，其中提供了有关 C++、Java 和 C#的入门知识。这些附加课程假设读者已经学习了 C 语言。

1.3　编程前的准备工作

解决问题时，应采取一些特定的步骤。首先必须定义问题。如果不知道问题是什么，将无法找到解决方案。知道问题是什么之后便可以设计解决它的方案。有了方案后，您通常能够实现它。方案实现后，必须对结果进行测试，以确定问题是否得到解决。这种逻辑也适用于包含编程在内的许多其他领域。

创建 C 语言程序（或其他语言的计算机程序）时，应遵循下面类似的步骤：

1. 确定程序的目标；
2. 确定要使用什么样的方法来编写程序；
3. 创建程序，以解决问题；
4. 运行程序，以查看其结果。

目标（参见第 1 步）可能是编写一个字处理器或数据库程序。一个更为简单的目标是将您的姓名显示到屏幕上。如果没有目标，将无法编写程序，因此必须首先完成第 1 步。

第 2 步是确定要使用什么样的方法来编写程序。需要使用一个计算机程序来解决问题吗？需要跟踪哪些信息？将使用什么样的公式？在这一步中，应确定需要知道哪些信息，应以什么样的次序来实现解决方案。

例如，假设有人请您编写一个计算圆面积的程序。第 1 步已经完成，因为目标已经明确：计算圆的面积。第 2 步是确定要计算圆的面积，需要知道哪些信息。这里假设用户提供圆的半径，知道这些信息后，您便可以使用公式 $p*r^2$ 获得答案。至此，您已经获得所需的信息，因此可以进入到第 3 步和第 4 步，这两步被称为程序开发周期。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

1.4 程序开发周期

程序开发周期有其自己的步骤。第 1 步是使用编辑器创建一个包含源代码的磁盘文件；第 2 步是编译源代码以创建目标文件；第 3 步是链接编译后的代码，创建一个可执行文件；第 4 步是运行程序，看其运行方式是否与规划相符。

1.4.1 创建源代码

源代码是一系列的语句或命令，用于指示计算机执行您期望的任务。正如前面指出的，程序开发周期的第 1 步是在编辑器中输入源代码。例如，下面是一行 C 语言源代码：

```
printf("Hello, Mom!");
```

上述语句命令计算机将消息“Hello, Mom!”显示到屏幕上（现在，不要关心该语句是如何工作的）。

使用编辑器

大多数编译器都自带了编辑器，可用来输入源代码，然而有些编译器没有。请查看编译器的用户手册，确定它是否自带了编辑器。如果没有，可使用其他编辑器。

大多数计算机系统都包含了一个可用作编辑器的程序。如果您的操作系统为 Linux 或 UNIX，则可以使用诸如 ed、ex、edit、emacs 或 vi 等编辑器；如果您使用的是 Microsoft Windows，则可以使用 NotePad 或 WordPad；如果您的操作系统是 MS/DOS 5.0 或更高的版本，则可以使用 Edit；如果您使用的是 5.0 之前的 DOS 版本，可使用 Edlin；如果您使用的是 PC/DOS 6.0 或更高的版本，则可以使用 E；如果您使用的是 OS/2，则可以使用编辑器 E 和 EPM。

大多数字处理器都使用特殊的编码来格式化其文档，其他程序无法正确地读取这些编码。美国信息交换标准码（ASCII）指定了一种几乎任何程序（包括 C 语言）都能够使用的标准文本格式。很多字处理器（如 WordPerfect、Microsoft Word、WordPad 和 WordStar）都能够以 ASCII 格式（以文本文件而不是文档文件）存储源代码文件。要将字处理器文件存储为 ASCII 文件，请在保存文件时选择“ASCII”或“文本”选项。

如果您不想使用上述任何编辑器，可以购买其他的编辑器。市面上有专门为输入源代码而设计的软件包（包括商用的和共享软件）。



注意：要寻找其他的编辑器，可以查看本地的计算机商店或计算机邮购目录，另一种方法是查看计算机编程杂志中的广告。本书附带的光盘中包含的两个编译器也自带了编辑器，它们是 Bloodshed Dev-C++和 DJGPP。有关这些程序的更详细的信息，请查阅附录 G 和附带光盘。

保存源代码文件时，必须给它取名。文件名应指出程序的功能。另外，保存 C 语言程序的源代码文件时，还应使用扩展名.c。虽然可以给源代码文件取任何名称和扩展名，但.c 是公认的扩展名。

1.4.2 编译源代码

虽然您能够（至少在阅读完本书后能够）理解 C 语言源代码，但计算机不能。计算机只能识别或被称为机器语言的二进制指令。必须将源代码转换为机器语言，C 语言程序才能在计算机上运行。这种转换工作（程序开发的第 2 步）是由被称作编译器的程序完成的。编译器将源代码文件作为输入，并生成一个磁盘文件，该文件中包含了与源代码语句对应的机器语言指令。编译器创建的机器语言指令被称为目标代码，而包含它们的磁盘文件被称为目标文件。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com



> 注意：本书是按照 ANSI 标准 C 编写的，因此读者使用何种 C 语言编译器无关紧要，只要它遵循 ANSI 标准即可。并非所有的编译器都支持该标准。当前的 C 语言标准名为 ISO/IEC 9899:1999。本书提到该标准时，将不使用这个复杂的名称，而是使用 C-99。

每种编译器都有用于创建目标代码的命令。编译时，通常使用该命令来运行编译器，后面加上源代码文件的名称。下面是使用各种 DOS/Windows 编译器来编译源代码文件 radius.c 时，应执行的命令：

- Microsoft C：cl radius.c；
- Borland's Turbo C：tcc radius.c；
- Borland C：bcc radius.c。

要在 UNIX 机器上编译 radius.c，请使用下面的命令：

```
cc radius.c
```

在使用 GCC 编译器的机器上，请执行下面的命令：

```
gcc radius.c
```

如果您使用的是本书附带光盘中 GNU C/C++编译器，也应执行 gcc 命令。有关使用何种命令来运行编译器，请查看编译器的用户手册。

如果您使用的是图形集成开发环境，则编译更简单。在大多数图形环境中，可以通过选择“编译”图标或菜单选项来编译程序清单。编译代码后，便可以选择“运行”图标或相应的菜单选项来执行该程序。有关编译和运行程序的细节，请查看编译器的用户手册。本书附带光盘中的 Bloodshed Dev-C++便是一个图形开发环境，可用于 Microsoft Windows 环境中。有的图形开发环境有几乎可以用于任何平台。

编译后，便获得了一个目标文件。如果您查看编译目录或文件夹中的文件列表，将看到一个名称同源代码文件相同、但扩展名为.obj（而不是.c）的文件。扩展名为.obj 的文件是目标文件，供链接程序使用。在 Linux 或 UNIX 系统中，编译器创建的目标文件的扩展名为.o，而不是.obj。

1.4.3　链接以创建可执行文件

运行程序前，还需要完成另一个步骤。ANSI C 语言定义中包含一个函数库，其中包含预定义的函数的目标代码（已经编译过的代码）。预定义的函数包含编写好的 C 代码，由编译器软件包以可以直接使用的方式提供。

前面的范例中使用的 printf()函数便是一个库函数。这些库函数执行经常需要完成的任务，如在屏幕上显示信息以及读取磁盘文件中的数据等。如果您的程序使用了这样的函数（几乎所有的程序都需要使用这样的函数），则必须将编译源代码时生成的目标文件和函数库中的目标代码组合起来，生成最终的可执行程序（可执行指的是程序可以在计算机上运行）。这一过程被称为链接，是由链接程序完成的。

图 1.1 说明了从源代码到目标代码，再到可执行程序的过程。

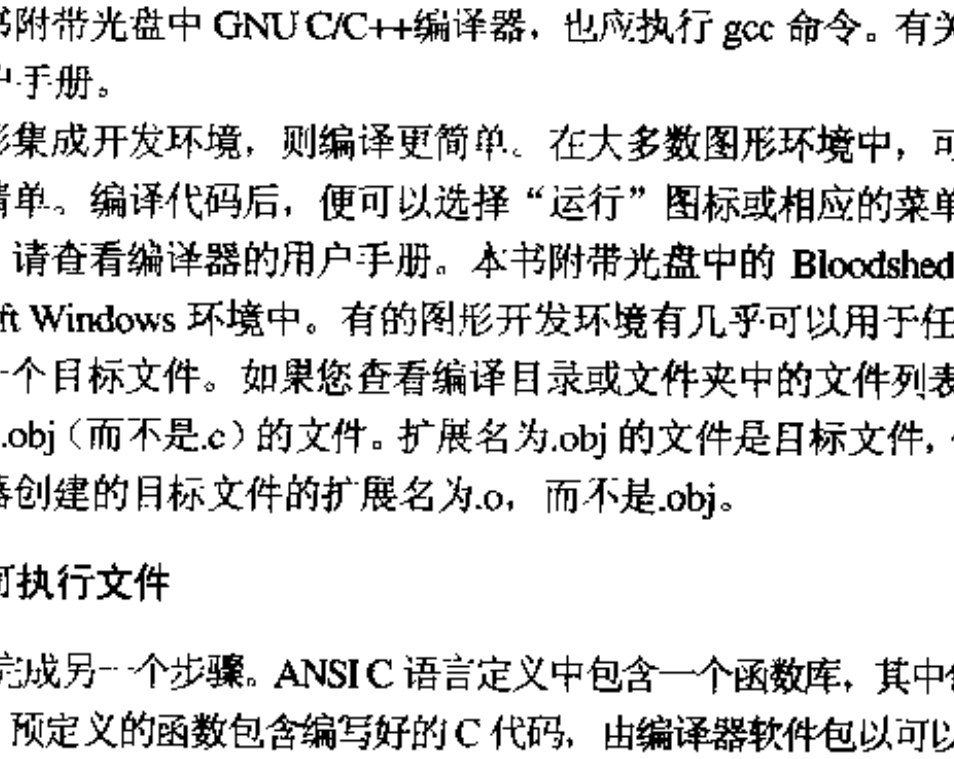


图 1.1　编译器将您编写的 C 语言源代码转换为目标代码，然后被链接程序转换为可执行文件

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

1.4.4 结束开发周期

将程序进行编译和链接，创建出可执行文件后，便可以在系统提示符下输入其名称（或像运行其他程序那样）运行它。如果运行程序时得到的结果与期望的不同，则需要回到第 1 步。您必须找出导致问题的原因，并在源代码中进行更正。修改源代码后，需要重新编译和链接程序，创建更正后的可执行文件。您将不断地沿这样的循环进行下去，直到程序的执行情况同期望的完全相符。

有关编译和链接，需要注意的最后一点是：虽然前面将编译和链接看作是两个独立的步骤，但很多编译器（如前面提到的 DOS 编译器）将它们在一步中完成。大多数图形开发环境提供了一个选项，让您设置是分别完成编译和连接还是一步完成。不管编译和连接工作是如何完成的，这两个过程都是独立的操作，即使使用一个命令来完成它们。

C 语言程序的开发周期如下：

1. 使用编辑器编写源代码。传统上，C 语言源代码文件的扩展名为.c（例如，myprog.c、database.c 等等）。

2. 使用编译器对程序进行编译。如果编译器没有发现任何错误，将生成一个目标文件。编译器生成的目标文件的扩展名为.obj 或.o，文件名与源代码文件相同（例如，myprog.c 被编译为 myprog.obj 或 myprog.o）。如果发现错误，编译器将报告。在这种情况下，您必须返回到第 1 步，在源代码中进行修改。

3. 使用链接程序对程序进行链接。如果没有发生错误，链接程序将生成一个可执行程序，该程序位于一个磁盘文件中，该文件的扩展名为.exe，文件名同目标文件相同（例如，链接 myprog.obj 时，生成的可执行文件为 myprog.exe）。

4. 执行程序。检查程序是否能够正确运行，如果不能，则返回到第 1 步，对源代码进行修改。

图 1.2 说明了上述程序开发步骤。除最简单的程序外，对于其他的所有程序，您都可能需要反复经过上述步骤才能完成程序的开发工作。即使是最有经验的程序员，也无法一次编写出完整的、没有任何错误的程序。由于您需要经历编辑-编译-链接-测试周期数次，因此熟悉这些工具（编辑器、编译器和链接程序）就显得至关重要。

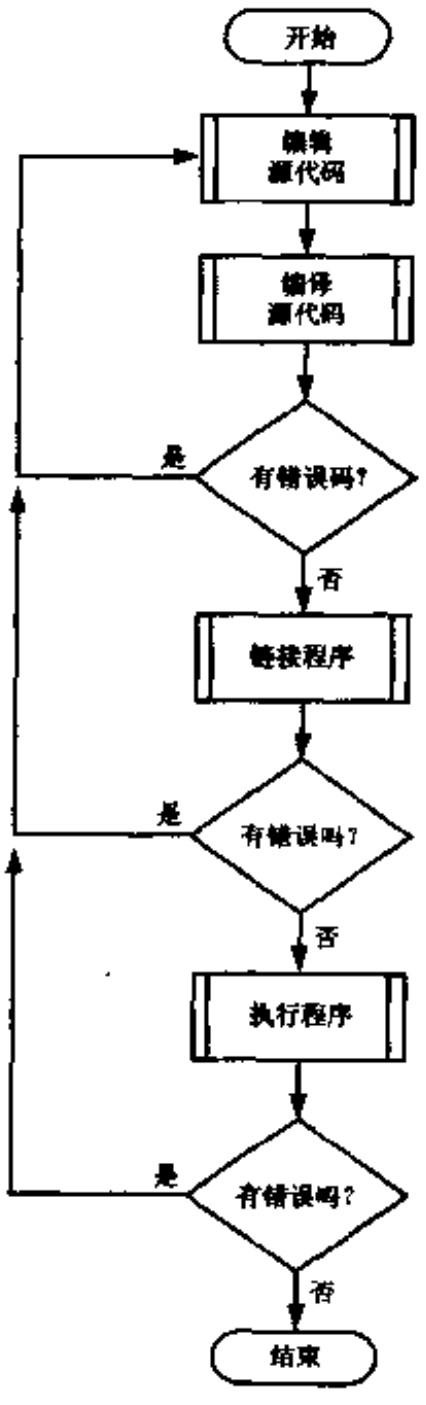


图 1.2 C 语言程序的开发步骤

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

1.5　第一个 C 语言程序

读者也许迫不及待地想编写第一个 C 语言程序。为帮助读者熟悉编译器，程序清单 1.1 包含一个小型程序，功能快速地完成。现在，读者也许无法理解其中的所有内容，但不用担心，尽管编写、编译并运行它。

这里的演示使用的是一个名为 hello.c 的程序，该程序只是将单词“Hello, World!”显示到屏幕上而已。该程序常被用来介绍 C 语言编程，很适合读者进行学习。程序清单 1.1 列出了 hello.c 的源代码。输入该程序清单时，请不要输入最左边的行号和冒号。

程序清单 1.1　　**HELLO.C**

```
1:  #include <stdio.h>
2:
3:  int main(void)
4:  {
5:      printf("Hello, World!\n");
6:      return 0;
7:  }
```

请务必按软件提供的安装说明安装编译器。无论您使用的是 Linux、DOS、UNIX、Windows 还是其他操作系统，请务必理解如何使用您所选择的编辑器和编译器。准备好编译器和编辑器后，请按下面的步骤输入、编译并执行 HELLO.C。

1.5.1　输入并编译 hello.c

要输入并编译 hello.c，请按下面的步骤进行：

1. 在要存储 C 语言程序的目录中启动编辑器。正如前面指出的，可以使用任何文本编辑器，但集成开发环境（IDE）自带的大多数编译器（如 Borland 的 Turbo C++和 Microsoft 的 Visual C++）都能让您以一种方便的设置来输入、编译和链接程序。请查看用户手册，了解编译器是否有 IDE。

2. 通过键盘输入 hello.c 中的源代码，内容应同程序清单 1.1 完全相同。换行时，请按 Enter 键。



注意：请不要输入行号和冒号，它们是为了便于引用而提供的。

3. 保存源代码，将其命名为 hello.c。

4. 通过列出目录中的文件列表，验证 hello.c 是否已被保存到磁盘中。文件列表中应包含文件 hello.c。

5. 编译并链接 hello.c。执行编译器用户手册指定的命令，您将看到一条消息，指出没有任何错误和警告。

6. 查看编译器消息。如果没有任何错误和警告，则说明一切正常。

如果输入的程序不正确，编译器将捕获您犯的错误，并显示错误消息。例如，如果将 printf 输入为 prntf，您将看到一条与下面类似的消息：

```
Error: undefined symbols:_prntf in hello.c (hello.OBJ)
```

7. 如果出现错误消息，请返回到第 2 步。在编辑器中打开 hello.c，将该文件内容同程序清单 1.1 进行仔细比较、修正，然后进入到第 3 步。

8. 至此，您的第一个 C 语言程序应编译好，可以执行了。此时如果显示所有名为 hello 的文件，将看到以下文件：

- hello.c：使用编辑器创建的源代码文件；
- hello.obj 或 hello.o：其中包含 hello.c 的目标代码；

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

- hello.exe：编译并链接 hello.c 时创建的可执行程序。

9. 要执行 hello.exe，只须执行 hello 命令即可，消息“Hello, World!”将显示在屏幕上。

祝贺您！您已经输入、编译并运行了您的第一个 C 语言程序。应该承认，hello.c 是一个很简单的程序，并不能完成任何有用的工作，但它开了一个头。事实上，当今大部分专家级 C 语言程序员都是以同样的方式——编译 HELLO.C——开始学习的。



注意：如果您使用的是本书附带光盘中提供的 Bloodshed Dev-C++编译器，请阅读附录 G，其中介绍了如何安装该编译器以及如何使用它来创建程序。该编译器可用于 Windows 95 或更高的版本中。

1. 编译错误

当编译器发现源代码中的某些内容无法编译时，将发生编译错误。拼写错误、印刷错误等都可能导致编译器停止工作。幸运的是，现代的大多数编译器不仅仅是停止工作，还会告诉您问题出在哪里！这使得查找和修改源代码中的错误更容易。

可以有意在前面输入的 hello.c 程序中加入错误来说明这一点。如果您完成了前面的范例，磁盘上将有 hello.c 的一个拷贝。在编辑器中打开该文件，然后将光标移到 printf()所在行的末尾，并删除最后的分号。此时，hello.c 的内容如程序清单 1.2 所示。

程序清单 1.2　　hello2.c：有错误的 hello.c

```
1:  #include <stdio.h>
2:
3:  int main(void)
4:  {
5:      printf("Hello, World!")
6:      return 0;
7:  }
```

然后保存该文件。现在可以编译它了，方法是执行编译器命令。由于刚才引入的错误，编译将无法完成。编译器将显示与下面类似的错误：

```
hello.c(6) : Error: ';' expected
```

上述消息包含三部分：

- hello.c：有错误的文件的名称；
- (6)：错误所在行的编号；
- Error: ';' expected：对错误的描述。

上述消息包含大量的信息，它指出：编译器发现 hello.c 第 6 行应该有一个分号，但没有。然而您知道，实际上是第 5 行遗漏了一个分号，这与消息所指出的不符。为何编译器报告第 6 行有错误，而实际上是第 5 行遗漏了分号呢？问题的答案在于，C 编译器并不关心行之间的换行符，printf()语句之后的分号也可以放在下一行的开头（虽然这样做容易引起混淆，不是一个好的编程习惯）。编译器遇到第 6 行的下一个命令后，才能确定遗漏了一个分号。因此，编译器指出第 6 行有错。

这指出了有关 C 语言编译器和错误消息的一个不可否认的事实。虽然编译器在检测和定位错误方面很聪明，但它并非爱因斯坦。您必须使用有关 C 语言的知识，对编译器的消息进行解读，以确定报告的错误的实际位置。通常，能够在编译器指出的行中找到错误，但如果找不到，则几乎总是在前一行。刚开始时，查找错误可能有些困难，但您很快便能得心应手。



注意：报告的错误可能随编译器而异。在大多数情况下，通过错误消息，您应该能够知道发生了什么错误或错误的位置。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

结束有关这个主题的讨论之前，请看另一个编译错误的例子。再次将 hello.c 装载到编辑器中，并做以下修改：

1. 在第 5 行的末尾加上分号；
2. 删除单词 Hello 前面的双引号。

保存文件，并再次编译该程序。这次，编译器将显示类似于下面的错误消息：

```
hello.c(5) : Error: undefined identifier 'Hello'
hello.c(7) : Lexical error: unterminated string
Lexical error: unterminated string
Lexical error: unterminated string

Fatal error: premature end of source file
```

第一条错误消息正确地指出了错误——第 5 行的 Hello。错误消息 undifined identifier 意味着编译器无法识别 Hello，因为它没有用引号括起。然而，其他 4 条错误消息呢？这些错误消息（现在无须关心它们的含义）表明，C 语言程序中的一个错误有时候可能导致多条错误消息。

我们从中得到的教训是：如果编译器报告了多个错误，而您只找到一个，请修复该错误并重新编译。您可能发现，只须修改一个地方，程序便没有任何错误，并能通过编译。

2. 链接程序的错误消息

链接程序错误比较少，通常是由于错误拼写 C 语言库函数的名称引起的。在这种情况下，将显示错误消息 Error: undefined symbols:，后面为拼错的名称（名称前有一个下划线）。更正拼写后，问题将得到解决。

1.6 总　结

阅读今天的课程后，读者将相信，选择 C 作为编程语言是一种明智的选择。C 语言融功能强大、流行和可移植性于一体，是其他语言所无法比拟的。这些因素同 C 语言与面向对象语言 C++、Java、C#的紧密关系一道，使得 C 语言是无与伦比的。

今天的课程介绍了编写 C 语言程序的各个步骤——程序开发过程。现在读者应对编辑-编译-链接-测试周期以及其中每一步使用的工具了如指掌。

在程序开发中，错误是不可避免的。C 语言编译器能够发现源代码中的错误，并显示错误消息——指出错误的性质和位置。根据这些信息，可以对源代码进行编辑，更正其中的错误。然而请切记，编译器并不能总是准确地指出错误的性质和位置。有时候，需要运用有关 C 语言的知识，找出导致错误消息的真正原因。

1.7 问与答

问：如果要将自己编写的程序提供给别人，应提供哪些文件？

答：C 语言的优点之一是，它是一种编译型语言。这意味着对源代码进行编译后，将得到一个可执行程序——一个独立的程序。将 hello 提供给所有有计算机的朋友是完全可能的，您只须将可执行程序 hello.exe 提供给他们即可。他们不需要源代码文件 hello.c 和目标文件 hello.obj，也不需要 C 编译器。但获得可执行程序的人必须拥有和您同类型的机器，如 PC、Macintosh、Linux 机器等。

问：创建可执行文件后，还需要保留源代码文件（.c）和目标文件（.obj）吗？

答：如果删除源代码文件，则以后将无法修改程序，因此应该保留该文件。目标文件的情况则不同，保留目标文件是有原因的，但这超出了您现在应该考虑的范围。就现在而言，一旦删除可执行文件后，便可以将目标文件删除。如果需要目标文件，可以重新编译源代码文件。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

大多数集成开发环境都会创建除源代码文件（.c）、目标文件（.obj 或.o）和可执行文件之外的其他文件。只要保留了源代码文件（.c），便可以重新创建其他文件。

问：如果编译器自带了编辑器，必须使用该编辑器吗？

答：完全可以不用。您可以使用任何编辑器，只要它能够以文本格式保存源代码。如果编译器自带了编辑器，应尽可能使用它。如果您要使用其他的编辑器，当然，也可以。作者使用的是一个单独购买的编辑器，虽然所有的编译器都自带了编辑器。编译器自带的编辑器越来越好，其中的一些能够自动格式化 C 语言代码，其他的一些则使用不同的颜色来显示源代码文件的不同部分，使得查找错误更容易。

问：如果只有 C++编译器，而没有 C 编译器，该如何办？

答：正如今天的课程中指出的，C++是 C 语言的超集。这意味着可以使用 C++编译器来编译 C 程序。大多数人在 Windows 环境下使用 Microsoft's Visual C++编译其 C 程序，在 Linux 和 UNIX 环境下，则使用 GNU 的编译器。本书附带的光盘中的编译器能够编译 C 程序和 C++程序。

问：可以忽略警告消息吗？

答：有些警告并不会影响程序的运行，但有些会。编译器显示警告消息表明有什么地方不正确。大多数编译器都允许用户设置警告等级。通过设置警告等级，可以只显示最严重的警告或显示所有的警告（包括最微不足道的）。有些编译器甚至提供了各种中间等级。应查看程序的每个警告，并对其做出判断。程序最好没有任何警告和错误（有错误时，编译器将不会创建可执行文件）。

1.8 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。在阅读下一课时之前，应尽可能理解这些小测验和练习的答案，答案见附录 F。

1.8.1 小测验

1. 指出 C 是首选编程语言的三个原因。
2. 编译器的功能是什么？
3. 程序开发周期中包含哪些步骤？
4. 使用您自己的编译器编译程序 program1.c 时，应执行什么命令？
5. 在您的编译器中，完成编译和链接工作只需一个命令，还是需要分别执行命令？
6. C 语言源代码文件应使用什么扩展名？
7. FILENAME.TXT 是一个合法的 C 语言源代码文件名吗？
8. 如果执行编译后的程序时，其工作方式与您期望的不同，应如何做？
9. 机器语言是什么？
10. 链接程序有何功能？

1.8.2 练习

1. 使用您的文本编辑器查看程序清单 1.1 创建的目标文件。目标文件像源代码文件吗（退出编辑器时，请不要保存该文件）？

2. 输入并编译下面的程序。该程序有何功能（请不要输入行号和冒号）？

```
1:  #include <stdio.h>
2:
3:  int radius, area;
4:
5:  int main( void )
6:  {
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
7:    printf( "Enter radius (i.e. 10): " );
8:    scanf( "%d", &radius );
9:    area = (int) (3.14159 * radius * radius);
10:   printf( "\n\nArea = %d\n", area );
11:   return 0;
12: }
```

3. 输入并编译下面的程序。该程序有何功能?

```
1: #include <stdio.h>
2:
3: int x, y;
4:
5: int main( void )
6: {
7:    for ( x = 0; x < 10; x++, printf( "\n" ) )
8:       for ( y = 0; y < 10; y++ )
9:          printf( "X" );
10:
11:   return 0;
12: }
```

4. 排错：下面的程序有问题。请输入并编译该程序。哪些行导致错误消息?

```
1: #include <stdio.h>
2:
3: int main( void );
4: {
5:    printf( "Keep looking!" );
6:    printf("you\'u find it!\n");
7:    return 0;
8: }
```

5. 排错：下面的程序有问题。请输入并编译该程序。哪些行导致错误消息?

```
1: #include <stdio.h>
2:
3: int main( vcid )
4: {
5:    printf( "This is a program with a " );
6:    do_it( "problem!");
7:    return 0;
8: }
```

6. 对练习 3 中的程序做如下修改，并重新编译和运行该程序。现在该程序有何功能?

```
9: printf( "%c", 1 );
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

TYPE & RUN 1　打印程序清单

本书包含大量的 TYPE & RUN（录入并运行），其中提供的程序清单比各章中的稍长，旨在让读者能够输入并运行它们。这些程序清单中可能包含本书没有介绍的内容。

这些程序通常执行一些有趣或实用的功能。例如，这里的程序名为 print_it。该程序除了打印源代码外，还打印行号，就像本书包含的程序清单那样。该程序可用来打印本书中其他的程序清单。

建议读者输入并运行这些程序后，花一些时间对其中的代码进行试验。对程序进行修改，然后重新编译并运行它们，看看会出现什么情况。作者不打算对这些代码的工作原理进行解释，而只说明其功能。但请不用着急，当您阅读完本书后，将能够理解这些程序清单中的所有内容。与此同时，您将有机会输入并运行一些更有趣或更实用的程序清单。

第一个 TYPE & RUN

输入并编译下面的程序。如果发现错误，请检查输入的程序是否正确。

该程序的用法是，执行 print_it filename.ext，其中 filename.ext 是源代码文件的名称（包括扩展名）。该程序在程序清单中加入了行号（不要被该程序的长度所吓倒，这里提供它旨在让您能够对程序的打印输出同本书中的程序清单进行比较，而并不要求您理解该程序）。

程序清单 T&R 1　print_it.c

```
1:  /* print_it.c-This program prints a listing with line numbers! */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:
5:  void do_heading(char *filename);
6:
7:  int line = 0, page = 0;
8:
9:  int main( int argv, char *argc[] )
10: {
11:   char buffer[256];
12:   FILE *fp;
13:
14:   if( argv < 2 )
15:   {
16:      fprintf(stderr, "\nProper Usage is: " );
17:      fprintf(stderr, "\n\nprint_it filename.ext\n" );
18:      return(1);
19:   }
20:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
21:   if (( fp = fopen( argc[1], "r" )) != NULL )
22:   {
23:       fprintf( stderr, "Error opening file, %s!", argc[1]);
24:       return(1);
25:   }
26:
27:   page = 0;
28:   line = 1;
29:   do_heading( argc[1]);
30:
31:   while( fgets( buffer, 256, fp ) != NULL )
32:   {
33:     if( line % 55 == 0 )
34:       do_heading( argc[1] );
35:
36:       fprintf( stdprn, "%4d:\t%s", line++, buffer );
37:   }
38:
39:   fprintf( stdprn, "\f" );
40:   fclose(fp);
41:   return 0;
42: }
43:
44: void do_heading( char *filename )
45: {
46:    page++;
47:
48:    if ( page > 1)
49:       fprintf( stdprn, "\f" );
50:
51:    fprintf( stdprn, "Page: %d, %s\n\n", page, filename );
52: }
```



注意：该程序清单使用了一个很多 PC 编译器中都可用的值，但并非所有其他的编译器都有这样的值。虽然 stdout 是一个 ANSI 定义的值，但 stdprn 不是。您必须查看您的编译器中有关将输出发送到打印机的细节。
避开这种问题的一种方法是，将 stdprn 语句修改为 stdout 语句。这将导致输出被发送到屏幕。使用操作系统的重定向功能（如果使用的 UNIX 或 Linux，请使用管道功能），可以将输出从屏幕重定向到打印机。

第 14 天的课程将介绍更多有关该程序的工作原理的知识。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 2 天课程　C 语言程序的组成部分

所有的 C 语言程序都是由多个部分通过特定的方式组合而成的。本书的大部分篇幅都旨在介绍程序的不同组成部分及其使用方法。为帮助说明 C 语言程序的整体情况，应首先来看一个完整的（虽然简短）C 语言程序，并指出其各个组成部分。今天您将学习以下内容：

- 一个简短的 C 语言程序及其组成部分。
- 每个组成部分的功能。
- 如何编译并运行范例程序。

2.1　一个简短的 C 语言程序

程序清单 2.1 列出了 multiply.c 的源代码。这个程序非常简单，它从键盘接受两个数字，并计算它们的乘积。现在，请不要过分关心有关程序工作原理的细节，重点是熟悉 C 语言程序的组成部分，以便能够更好地理解本书后面的程序清单。

介绍范例程序之前，先介绍什么是函数，因为函数是 C 语言编程的核心。函数是一段独立的程序代码，它执行特定的任务，并被指定了名称。通过引用函数的名称，程序能够执行函数中的代码。程序还能够将信息（称为参数）传递给函数，而函数则可以将信息返回给函数的主要部分。C 函数有两种：库函数和用户定义的函数，前者位于 C 编译器软件包中，而后者是由程序员创建的。本书将介绍这两种函数。

注意，与本书中的所有其他程序清单一样，程序清单 2.1 中的行号并非程序的组成部分。提供它们只是为了方便标识，因此请不要输入。

在本书附带的光盘中可以找到程序清单 2.1，它位于目录 Day02 中。

程序清单 2.1　multiply.c：计算两个数的乘积

```
1:  /* Program to calculate the product of two numbers. */
2:  #include <stdio.h>
3:
4:  int val1, val2, val3;
5:
6:  int product(int x, int y);
7:
8:  int main( void )
9:  {
10:    /* Get the first number */
11:    printf("Enter a number between 1 and 100: ");
12:    scanf("%d", &val1);
13:
14:    /* Get the second number */
15:    printf("Enter another number between 1 and 100: ");
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
16:     scanf("%d", &val2);
17:
18:     /* Calculate and display the product */
19:     val3 = product(val1, val2);
20:     printf ("%d times %d = %d\n", val1, val2, val3);
21:
22:     return 0;
23: }
24:
25: /* Function returns the product of the two values provided */
26: int product(int x, int y)
27: {
28:     return (x * y);
29: }
```

该程序的运行情况如下：

```
Enter a number between 1 and 100: 35
Enter another number between 1 and 100: 23

35 times 23 = 805
```

2.2　程序的组成部分

接下来的几个小节描述了上述范例程序的各个组成部分，其中指出了行号，以便读者能够迅速找到被讨论的程序组成部分。

2.2.1　main()函数（第 8～23 行）

在所有可执行的 C 语言程序中，唯一必不可少的部分是 main()函数。最简单的情况下，main 函数由名称 main、包含 void 的一对圆括号（(void)）和一对花括号（{}）组成。对于大部分编译器，可以省略单词 void，程序仍能够正常运行。ANSI 标准规定，应该包含单词 void，以便知道没有给 main 函数传递任何信息。

在花括号中，包含的是组成程序主体的语句。通常情况下，程序从 main 函数中的第一条语句开始执行，到 main 函数中的最后一条语句结束。根据 ANSI 标准，必须包含的唯一一条语句是本范例中第 22 行的 return 语句。

2.2.2　#include 编译指令（第 2 行）

编译指令#include 命令 C 编译器，在编译时将一个包含文件的内容添加到程序中。包含文件是一个独立的磁盘文件，其中包含可被程序或编译器使用的信息。编译器提供了多个这样的文件（有时被称为头文件）。通常不需要修改这些文件中的信息，这也是将其独立于源代码文件的原因。包含文件的扩展名总是为.h（如 studio.h）。

使用编译指令#include 来命令编译器，在编译时将指定的包含文件添加到程序中。在程序清单 2.1 中，编译指令#include 的含义是，加入文件 studio.h 的内容。在 C 语言程序中，几乎总是要包含一个或多个包含文件。有关包含文件的更详细的信息，请参阅第 21 天的课程。

2.2.3　变量定义（第 4 行）

变量是给用于存储信息的内存单元赋于名称。在程序执行期间，程序使用变量来存储各种信息。在 C 语言中，使用变量之前必须定义它。变量定义将变量的名称以及变量要存储的信息类型告知编译器。在该范例

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序中，第 4 行的定义 int val1, val2, val3;定义了三个变量，它们分别名为 val1、val2 和 val3，都用于存储一个整型值。有关变量和变量定义的更详细的信息，请参阅第 3 天的课程。

2.2.4 函数原型（第 6 行）

函数原型将程序中包含的函数的名称和参数告知编译器，位于函数被使用之前的位置。函数原型不同于函数定义，后者包含组成函数的实际语句（有关函数定义，将在今天课程的后面做更详细的讨论）。

2.2.5 程序语句（第 11、12、15、16、19、20、22 和 28 行）

C 语言程序的实际工作是由其语句完成的。C 语句将信息显示到屏幕上、读取键盘输入、执行数学运算、调用函数、读取磁盘文件以及程序需要执行的其他操作。本书的大部分篇幅用于介绍各种 C 语句。就现在而言，您只需记住，在源代码中，每条 C 语句通常占一行，并且总是以分号结尾。接下来的几节将简要地介绍 multiply.c 中的语句。

1. printf()语句

printf()语句（第 11、15 和 20 行）是一个库函数，它将信息显示到屏幕上。printf()语句能够显示简单的文本消息（如第 11 和 15 行），也可以显示一条消息和一个或多个变量的值（如第 20 行）。

2. scanf()语句

scanf()语句（第 12 和 16 行）是另一个库函数，它读取键盘输入，并将输入赋给一个或多个变量。

第 19 行的程序语句调用函数 product()，换句话说，它执行函数 product()中的程序语句。它还将参数 val1 和 val2 传递给这个函数。函数 product()中的语句执行完毕后，product()将一个值返回给程序，这个值被存储在变量 val3 中。

3. return 语句

第 22 行和 28 行包含 return 语句。第 28 行的 return 语句是函数 product()的一部分，它计算变量 x 和 y 的乘积，并将结果返回给调用 product()的程序。第 22 行的 return 语句在程序结束之前，将 0 返回给操作系统。

2.2.6 函数定义（第 26～29 行）

函数是一个独立的、自主式代码段，用于完成特定的任务。每个函数都有名称，函数中的代码是通过程序语句中包含函数的名称来执行的，这被称为调用函数。

第 26～29 行的函数 product()是一个用户自定义的函数。顾名思义，用户自定义的函数是由程序员在程序开发过程中编写的。这个位于第 26～29 行的函数很简单，它只是将两个值相乘，并将结果返回给调用它的程序。在第 5 天的课程中，您将知道正确使用函数是良好 C 语言编程习惯的重要组成部分。

注意，在实际的 C 语言程序中，可能不会使用函数来完成诸如计算两个数的乘积这样简单的任务。这里这样做只是出于演示的目的。

C 语言还包含库函数，库函数位于 C 编译器软件包中。库函数执行程序所需的大部分常见任务，如屏幕、键盘、磁盘输入/输出。在上述范例程序中，printf()和 scanf()都是库函数。

2.2.7 程序注释（第 1、10、14、18 和 25 行）

程序中以/*开始，并以*/结束的部分被称为注释。编译器忽略所有的注释，因此注释对程序的运行没有任何影响。可以在注释中添加任何内容，而不会改变程序的运行方式。注释可以占一行、多行或一行的一部分。下面是三个注释的例子：

```
/* A single-line comment */
int a,b,c; /* A partial-line comment */
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
/* a comment
spanning
multiple lines */
```

注释不能嵌套，嵌套注释指的是位于另一个注释中的注释，大多数编译器不允许下面这样的注释：

```
/*
/* Nested comment */
*/
```

有些编译器确实允许嵌套注释。虽然这种特性很有诱惑力，但应避免使用。因为 C 语言的优势之一是可移植性，使用诸如嵌套注释这样的特性将限制代码的移植性。嵌套注释还可能导致难以发现的问题。

很多初级程序员将注释视为多余和浪费时间。这完全是错误的！当您编写代码时，可能对程序的逻辑非常清楚，然而，随着程序逐渐变大、变复杂或者当您需要修改半年前编写的程序时，您将发现注释的价值是无法衡量的。应养成使用注释来说明编程结构和逻辑的习惯。

最新的 ANSI 标准新增了使用单行注释的功能。在 C++和 Java 中，早已允许使用单行注释，因此在 C 语言中实现这种特性是自然而然的。

单行注释使用双斜杠来标识注释。下面是两个这样的例子：

```
// This entire line is a comment
int x; // Comment starts with slashes.
```

两个斜杠指出该行中余下的内容为注释。 ANSI C-99 标准将这种特性加入了 C 语言中。

2.2.8　使用花括号（第 9、23、27 和 29 行）

使用花括号（{}）将组成每个 C 函数（包括 main()函数）的程序行括起。用花括号括起的一条或多条语句被称为代码块。正如您将在本书后面看到的，在 C 语言中，代码块的用途很多。

应　该	不 应 该
应在程序的源代码中添加大量的注释，尤其是对于这样的语句和函数，即您或以后需要修改该程序的人可能无法理解它。 应养成有帮助的编程风格。晦涩的编程风格是有害的，冗长的编程风格可能导致您用于编写注释的时间多于编程的时间。	不应给已经很清晰的语句添加注释，例如，下面的内容可能有些小题大做，尤其是当您已经熟悉 printf()函数及其工作原理时： `/* The following prints Hello World!` `on the screen */` `printf("Hello World!");`

2.2.9　运行程序

请输入、编译并运行 multiply.c，这让您进一步练习使用编辑器和编译器。第 1 天的课程介绍过，完成这项工作的步骤如下：

1. 将编程目录设置为当前目录；
2. 启动编辑器；
3. 按程序清单 2.1 输入 mulutiply.c，但不要输入其中的行号和冒号；
4. 将程序文件存盘；
5. 通过执行合适的命令，编译和链接该程序。如果没有出现错误消息，则可以在命令提示符下执行 multiply 命令来运行该程序；
6. 如果出现错误消息，则返回到第 2 步，并更正错误。



注意：如果您使用的是 Dev-C++，请参阅附录 G，了解如何输入和编译程序。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

2.2.10 有关精度的说明

计算机快速而准确，但也非常呆板。它无法更正最简单的错误，只是接受您输入的内容，而不管您的实际意思是什么。

C 语言源代码也是如此。程序中简单的拼写错误都可能导致 C 编译器停止工作，甚至崩溃。幸运的是，虽然编译器无法更正您的错误（您不可避免地会犯错误，所有人都如此），但能够识别并报告错误（在昨天的课程中，介绍了编译器如何报告错误消息以及如何解释它们）。

2.3 重温程序的组成部分

了解程序的各个组成部分后，您将能够识别任何程序的组成部分。请看程序清单 2.2，看看您能否识别其不同的组成部分。

程序清单 2.2　　list_it.c：显示程序清单

```
1:  /* list_it.c__This program displays a listing with line numbers! */
2:  #include <stdio.h>
3:  #include <stdlib.h>
4:
5:  void display_usage(void);
6:  int line;
7:
8:  int main( int argc, char *argv[] )
9:  {
10:    char buffer[256];
11:    FILE *fp;
12:
13:    if( argc < 2 )
14:    {
15:       display_usage();
16:       return 1;
17:    }
18:
19:    if (( fp = fopen( argv[1], "r" )) == NULL )
20:    {
21:       fprintf( stderr, "Error opening file, %s!", argv[1] );
22:       return(1);
23:    }
24:
25:    line = 1;
26:
27:    while( fgets( buffer, 256, fp ) != NULL )
28:       fprintf( stdout, "%4d:\t%s", line++, buffer );
29:
30:    fclose(fp);
31:    return 0;
32: }
33:
34: void display_usage(void)
35: {
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
36:      fprintf(stderr, "\nProper Usage is: " );
37:      fprintf(stderr, "\n\nlist_it filename.ext\n" );
38: }
```

该程序的运行情况如下：

```
C:\>list_it list_it.c
1:   /* list_it.c - This program displays a listing with line numbers!*/
2:   #include <stdio.h>
3:   #include <stdlib.h>
4:
5:   void display_usage(void);
6:   int line;
7:
8:   int main( int argc, char *argv[] )
9:   {
10:      char buffer[256];
11:      FILE *fp;
12:
13:      if( argc < 2 )
14:      {
15:          display_usage();
16:          return 1;
17:      }
18:
19:      if (( fp = fopen( argv[1], "r" )) == NULL )
20:      {
21:          fprintf( stderr, "Error opening file, %s!", argv[1] );
22:          return(1);
23:      }
24:
25:      line = 1;
26:
27:      while( fgets( buffer, 256, fp ) != NULL )
28:        fprintf( stdout, "%4d:\t%s", line++, buffer );
29:
30:      fclose(fp);
31:      return 0;
32: }
33:
34: void display_usage(void)
35: {
36:      fprintf(stderr, "\nProper Usage is: " );
37:      fprintf(stderr, "\n\nlist_it filename.ext\n" );
38: }
```

分析：程序清单 2.2 中的程序 list_it.c 显示您保存的 C 程序清单。程序清单将被显示在屏幕上，同时添加了行号。

看看该程序清单的各个组成部分。必不可少的 main()函数位于第 8～32 行；第 2 和第 3 行分别是#include 编译指令；第 6、10 和 11 行是变量定义；第 5 行是一个函数原型——void display_usage(void)。该程序包含很多语句（第 13、15、16、19、21、22、25、27、28、30、31、36 和 37 行）。第 34～38 行是 display_usage() 的函数定义。整个程序中的代码块由花括号括起。最后，只有第 1 行包含注释。在大多数程序中，可能包含

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

多行注释。

list_it.c 调用了很多函数，但只调用了一个用户定义的函数——display_usage()。该程序使用的库函数有 fopen()（第 19 行）、fprintf()（第 21、28、36 和 37 行）、fgets()（第 27 行）和 fclose()（第 30 行）。本书的其他部分将更详细地介绍这些库函数。

2.4 总 结

今天的课程不长，但很重要，因为它介绍了 C 语言程序的主要组成部分。您知道，每个 C 语言程序唯一必不可少的部分是 main()函数；程序的实际工作是由程序语句完成的，语句命令计算机执行所需的任务。另外，今天的课程还介绍了变量和变量定义，同时您还学会了如何在源代码中使用注释。

除了 main()函数外，C 语言程序还可以使用两种辅助函数：库函数和用户自定义的函数，前者是编译器软件包提供的，而后者是由程序员创建的。接下来几天的课程将更详细地介绍今天所提到的 C 语言程序的许多组成部分。

2.5 问与答

问：注释对程序有何影响？

答：注释是供程序员查看的。当编译器将源代码转换为目标代码时，将忽略其中的注释和空白，这意味着它们对可执行程序没有任何影响。包含大量注释的程序在执行时，速度将和几乎没有注释的程序一样快。注释确实会加大源代码文件，但通常这是无关紧要的。总之，您应使用注释和空白来提高源代码的可读性和可维护性。

问：语句和代码块之间有何区别？

答：代码块是一组用花括号（{}）括起的语句。在可以使用语句的大部分地方，都可以使用代码块。

问：如何获悉哪些库函数可用？

答：很多编译器都带有一个专门用于说明库函数的用户手册，这些函数通常按字母顺序排列。另一种获悉有哪些库函数可用的方法是，购买一本专门介绍库函数的图书。附录 E 列出了很多库函数。当您更深入地了解 C 语言后，阅读该附录以防重新编写库函数将是一个不错的主意（做重复的工作没有任何意义）。

2.6 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

2.6.1 小测验

1. 用花括号括起的一组语句叫什么？
2. 在每个 C 语言程序中都必不可少的是哪个组成部分？
3. 如何添加程序注释？为何要使用注释？
4. 函数是什么？
5. C 语言提供了哪两种函数？它们之间有何区别？
6. 编译指令#include 有何用途？
7. 注释可以嵌套吗？
8. 注释可以超过一行吗？
9. 包含文件又叫什么？

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

10. 包含文件是什么?

2.6.2 练习

1. 编写一个最简单的程序。
2. 请看下面的程序:

```
1:  /* ex02-02.c */
2:  #include <stdio.h>
3:
4:  void display_line(void);
5:
6:  int main(void)
7:  {
8:     display_line();
9:     printf("\n Teach Yourself C In 21 Days!\n");
10:     display_line();
11:
12:     return 0;
13: }
14:
15: /* print asterisk line */
16: void display_line(void)
17: {
18:    int counter;
19:
20:    for( counter = 0; counter < 30; counter++ )
21:       printf("*" );
22: }
23: /* end of program */
```

a. 哪些行包含语句?

b. 哪些行包含变量定义?

c. 哪些行包含函数原型?

d. 哪些行包含函数定义?

e. 哪些行包含注释?

3. 编写一个注释。
4. 下面的程序有何功能（请输入、编译并运行它）?

```
1:  /* ex02-04.c */
2:  #include <stdio.h>
3:
4:  int main(void)
5:  {
6:     int ctr;
7:
8:     for( ctr = 65; ctr < 91; ctr++ )
9:        printf("%c", ctr );
10:
11:    return 0;
12: }
13: /* end of program */
```

5. 下面的程序有何功能（请输入、编译并运行它）?

```
1:  /* ex02-05.c */
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
2:  #include <stdio.h>
3:  #include <string.h>
4:  int main(void)
5:  {
6:      char buffer[256];
7:
8:      printf( "Enter your name and press <Enter>:\n");
9:      gets( buffer );
10:
11:      printf( "\nYour name has %d characters and spaces!",
12                      strlen( buffer ));
13:
14:     return 0;
15: }
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 3 天课程　存储信息：变量和常量

计算机程序通常使用不同类型的数据，因此需要采用某种方式来存储被使用的值。这些值可能是数字，也可能是字符。C 语言有两种存储数值的方式：变量和常量，其中每一种方式又有多种选项。变量是一个数据存储位置，其值在程序执行期间可能发生变化；而常量的值是固定的，不能修改。今天将介绍以下内容：

- 如何使用变量来存储信息；
- 高效地存储不同类型数值的方式；
- 字符和数值之间的异同；
- 如何声明和初始化变量；
- C 语言中的两种数值常量。

学习变量之前，读者必须了解一些有关计算机内存的知识。

3.1　计算机内存

如果读者已经了解计算机内存的运行方式，可以跳过本节；否则，请阅读本节。理解计算机内存及其工作原理将有助于读者更好地理解 C 语言编程的某些方面。

计算机运行时，使用随机存储器（RAM）存储信息。RAM 通常位于计算机的内部，它是易失的，即必要时将被擦除，并被替换为新的信息；同时仅当计算机开启时，其中的信息才可用，一旦计算机被关闭，其中的信息将丢失。

每一台计算机都安装了一定数量的 RAM。系统中的 RAM 量通常以 MB（兆字节）为单位，如 2MB、4MB、8MB、32MB 等。1MB 的内存为 1024KB，而 1KB 等于 1024 字节。因此 4MB 内存的系统拥有 4×1024（4096）KB RAM，这相当于 4194304（4096×1024）字节。

字节是计算机数据存储空间的基本单位，第 20 天的课程将更详细地介绍。表 3.1 列出了存储一些数据所需的字节数。

表 3.1　存储数据所需的内存空间

数　据	所需的字节数
字母 x	1
数字 500	2
数字 241.105	4
短语 Sams Teach Yourself C	22
一页内容	大约 3000

计算机中的 RAM 是依次逐字节排列的。每个字节的内存都有一个唯一的地址，用于标识该字节，该地址可用于将该字节同内存中的其他字节区分开来。内存的地址是依次指定的，最小值为 0，最大值取决于系统的内存容量。现在，读者无需关心地址，地址是由 C 编译器自动处理的。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

计算机的 RAM 是做什么的呢？它有多种用途，但作为程序员，只需关心的是用作数据存储空间。数据是 C 语言程序使用的信息。无论程序的功能是维护地址列表、监视股票行情、记录家庭预算，还是跟踪猪肚的价格，在程序运行期间，信息（姓名、股价、家庭开销、猪肚价格）都被保存在计算机的 RAM 中。

了解有关内存的一些具体细节后，便可以回到 C 编程技术，了解 C 语言是如何使用内存来存储信息的。

3.2 使用变量存储信息

变量是计算机内存中一个被命名的数据存储位置。在程序中使用变量名时，实际上引用的是存储在这里的数据。

3.2.1 变量名

要在 C 语言程序中使用变量，必须知道如何给变量命名。在 C 语言中，变量名必须遵循以下规则：

- 名称可以包含字母（a~z、A~Z）、数字（0~9）和下划线（_）；
- 第 1 个字符必须是字母；第 1 个字符也可以是下划线，但不推荐这样做；第一个字符不能是数字（0~9）；
- 大小写是有区别的。C 语言是区分大小写的，因此名称 count 和 Count 指的是不同的变量。
- C 语言关键字不能用作变量名。关键字是 C 语言的一个组成部分（有关关键字的完整列表，请参阅附录 B）。

下面是一些合法和非法的变量名：

- Percent：合法；
- y2x5_fg7h：合法；
- annual_profit：合法；
- _1990_tax：合法但不推荐；
- savings#account：非法，因为包含非法字符#；
- double：非法，因为这是一个 C 关键字；
- 4sale：非法，因为第一个字符不能是数字。

由于 C 语言是区分大小写的，因此名称 percent、PERCENT 和 Percent 将被视为不同的变量。C 程序员通常在变量名中使用小写字母，虽然这不是必须的。全部大写通常用于常量名（这将在今天课程的后面介绍）。

对于很多 C 编译器来说，变量名最多可包含 31 个字符（实际上可以包含更多的字符，但编译器只考虑前面的 31 个字符）。有了这种灵活性后，您可以创建反映将被存储的数据变量名。例如，计算贷款偿还的程序可以将最低利率存储在一个名为 interest_rate 的变量中，该变量名清晰地指出了其用途。当然，您也可以创建名为 x，甚至 ozzy_osborne 的变量。这对于 C 编译器而言是无关紧要的，但当其他人查看源代码时，将无法知道其用途。虽然输入描述性的变量名可能需要更多的时间，由于这样可以使程序更为清晰，因此这样做是值得的。

对于用多个单词组合成的变量名，有很多命名约定。前面已经介绍了一种风格：interest_rate。使用下划线将变量名的单词分开，这样解释起来更为容易。另一种风格被称为驼峰表示法（camel notation），这种表示法不使用下划线，而是将每个单词的第一个字母大写。因此变量 interest_rate 可以被命名为 InterestRate。驼峰表示法正越来越流行，因为输入大写字母比输入下划线更容易。本书之所以使用下划线，是因为对于大多数人而言，它更易于阅读。读者可以自己决定要采用的风格。

应　该	不 应 该
应使用描述性的变量名。	不应在不必要的情况下，将下划线用做变量名的首字符。
应采用一个命名变量的风格，并一直坚持使用。	不要在不必要的情况下，将变量名的所有字符都大写。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

3.3　数值变量的类型

C 语言提供了多种数值变量类型。由于不同的数值需要的内存空间不同，对它们执行的数学运算也不同，因此您需要不同的变量类型。存储小型整数（如 1、199 和-8）时需要的内存较少，计算机对这样的数字执行数学运算时速度非常快；而大型整数和浮点数（如 123000000、3.14 或 0.000000871256）需要的存储空间更多，执行数学运算所需的时间更长。通过使用合适的变量类型，可以确保程序运行的效率尽可能高。

数值变量分为以下两大类：

- 整型变量：存储没有小数的数值（即整数），分两类：有符号整型变量和无符号整型变量，前者可以存储正值和负值，而后者只能存储正值（和 0）。
- 浮点型变量：存储带小数的值（即实数）。

上述两大类又分别包含两种或多种变量类型。表 3.2 对此进行了总结，其中还指出了存储每种变量所需的内存量（单位为字节）。

表 3.2　数值数据类型

变量类型	关键字	所需内存（字节）	取值范围
字符	char	1	-128~127
短整型	short	2	-32767~32767
整型	int	4	-2147483647~2147438647
长整型	long	4	-2147483647~2147438647
特长整型	long long	8	-9223372036854775807~9223372036854775807
无符号字符	unsigned char	1	0~255
无符号短整型	unsigned short	2	0~65535
无符号整型	unsigned int	4	0~4294967295
无符号长整型	unsigned long	4	0~4294967295
无符号特长整型	unsigned long long	8	0~18446744073709551615
单精度浮点数	float	4	1.2E-38~3.4E38[1]
双精度浮点数	double	8	2.2E-308~1.8E308[2]

[1] 大概范围，精度为 7 位

[2] 大概范围，精度为 19 位



注意：大概范围指的是指定变量能够存储的最大和最小值（由于空间限制，无法准确列出这些变量的取值范围）。精度指的是变量被存储时的准确程度。例如，如果 1/3 的值为 0.3333...，将其存储在精度为 7 的变量中时，将存储 7 个 3。



警告：不支持 C-99 标准的编译器可能不支持数据类型 long long 和 unsigned long long。

从表 3.2 可知，数据类型 int 和 short 是相同的。为何需要这两种数据类型呢？在 32 位的 Intel 系统（PC）中，int 和 short 是相同的，但在其他系统中，它们可能不同。例如，在 VAX 系统中，short 和 int 的长度并不同，int 占用 4 个字节，而 short 占用两个字节。别忘了，C 语言是一种非常灵活、可移植的语言，因此提供了这两种类型的关键字。在 PC 上，int 和 short 可以互换。

将整型变量声明为有符号时，不需要使用特殊的关键字，默认情况下，整型变量为有符号的。但是，如果您愿意，也可以使用关键字 signed。表 3.2 列出的关键字用于声明变量，这将在下一节进行讨论。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 3.1 可以帮助您确定在特定的计算机上，变量占用的内存空间。如果您运行该程序时得到的输出与程序清单后面的输出不同，请不用感到大惊小怪。

程序清单 3.1　　sizeof.c：显示变量类型的大小

```
1:  /* sizeof.c-Program to tell the size of the C variable */
2:  /*          type in bytes */
3:
4:  #include <stdio.h>
5:
6:  int main(void)
7:  {
8:     printf("\nA char      is %d bytes", sizeof( char ));
9:     printf( "\nAn int      is %d bytes", sizeof( int ));
10:    printf( "\nA short     is %d bytes", sizeof( short ));
11:    printf( "\nA long      is %d bytes", sizeof( long ));
12:    printf( "\nA long long is %d bytes\n", sizeof( long long));
13:    printf( "\nAn unsigned char  is %d bytes", sizeof( unsigned char ));
14:    printf( "\nAn unsigned int   is %d bytes", sizeof( unsigned int ));
15:    printf( "\nAn unsigned short is %d bytes", sizeof( unsigned short ));
16:    printf( "\nAn unsigned long  is %d bytes", sizeof( unsigned long ));
17:    printf( "\nAn unsigned long long is %d bytes\n",
18:                                sizeof( unsigned long long));
19:    printf( "\nA float     is %d bytes", sizeof( float ));
20:    printf( "\nA double    is %d bytes\n", sizeof( double ));
21:    printf( "\nA long double is %d bytes\n", sizeof( long double ));
22:
23:    return 0;
24: }
```

该程序的输出如下：

```
A char      is 1 bytes
An int      is 4 bytes
A short     is 2 bytes
A long      is 4 bytes
A long long is 8 bytes

An unsigned char  is 1 bytes
An unsigned int   is 4 bytes
An unsigned short is 2 bytes
An unsigned long  is 4 bytes
An unsigned long long is 8 bytes

A float     is 4 bytes
A double    is 8 bytes
A long double is 12 bytes
```

分析：上述输出表明，程序清单 3.1 指出每种变量在您的计算机上占用多少字节的内存。如果您使用的是标准的 32 位的 PC，则输出将同表 3.2 列出的值相同。

不用为理解程序的不同部分而操心。虽然其中有一些新内容，如 sizeof，但其他内容是以前见过的。第 1 和 2 行是注释，指出了程序的名称，并对程序做了简要的描述。第 4 行包含了标准输入/输出头文件，以帮助在屏幕上打印信息。这个程序很简单，只包含一个函数——main()（第 7～24 行）。程序的主要部分位于第 8～

鸿鹄论坛 http://bbs.hh010.com

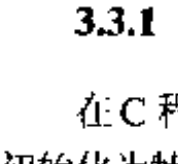
www.plcworld.cn

鸿鹄书院 http://book.hh010.com

21 行，其中每一行都打印一行文本，指出每种变量的大小，这是通过使用运算符 sizeof 完成的。第 23 行在程序结束之前将 0 返回给操作系统。

虽然前面指出过，数据类型的大小随计算机平台而异，但有些是确定的。下面的 5 点对任何计算机平台都是正确的：

- char 的大小为一个字节；
- short 的长度不会超过 int；
- int 的长度不会超过 long；
- unsigned 的长度等于 int；
- float 的长度不会超过 double。



注意：表 3.2 列出了通常用来标识各种变量的关键字，而表 3.3 列出了每种数据类型的全称。

从该表可知，short 和 long 类型是 int 类型的变体。大多数程序员不使用变量类型的全称，而使用其简称。

表 3.3　　数据类型的全称

全　称	常用的关键字
char	signed char
short	signed short int
int	signed int
long	signed long int
long long	signed long long int
unsigned char	unsigned char
unsigned short	unsigned short int
unsigned int	unsigned int
unsigned long	unsigned long int
unsigned long long	unsigned long long int

3.3.1　变量声明

在 C 程序中，使用变量之前必须声明它。变量声明将变量的名称和类型告诉编译器。声明还可能将变量初始化为特定的值。如果程序试图使用一个未经声明的变量，编译器将生成一条错误消息。变量声明的格式如下：

```
typename varname;
```

其中 typename 是变量的类型，必须为表 3.2 列出的关键字之一；varname 是变量的名称，必须遵循前面介绍的规则。可以在同一行中声明多个同一类型的变量，只需将变量名用逗号隔开即可：

```
int count, number, start;    /* three integer variables */
float percent, total;        /* two float variables */
```

第 12 天的课程将指出，变量声明在源代码中的位置至关重要，因为它影响程序能够以什么样的方式使用该变量。就现在而言，您可以将所有变量声明放在一起，并位于 main()函数之前。

3.3.2　typedef 关键字

关键字 typedef 用于给已有的数据类型指定一个新的名称。实际上，typedef 创建一个同义词，例如下面的语句：

```
typedef int integer;
```

将 integer 作为 int 的同义词。然后，您便可以使用 integer 来定义 int 变量，如下面的范例所示：

```
integer count;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

请注意：typedef 并不创建新的数据类型，而只是让您能够将不同的名称用于一个预定义的数据类型。Typedef 最常用于聚集数据类型，这将在第 11 天的课程中介绍。聚集数据类型是由多种数据类型组合而成的。

3.3.3 初始化变量

声明变量时，使指示了计算机为变量留出存储空间。然而，存储在该空间中的值——变量的值——并没有指定，它可能为零，也可能为一个随机的“垃圾”值。使用变量之前，一定要将其初始化为一个确定的值。可以在声明变量之后，使用赋值语句来初始化变量，如下面的范例所示：

```
int count;   /* Set aside storage space for count */
count = 0;   /* Store 0 in count */
```

注意，该语句使用了等号（=），这是 C 语言中的赋值运算符，将在第 4 天的课程中做进一步的讨论。现在，您需要明白的是，编程中的等号和代数中的等号并不完全相同。在代数中，下面的表达式表示“x 等于 12”：

```
x = 12
```

而在 C 语言中，其含义则完全不同，它指的是将 12 赋给变量 x。

也可以在声明时初始化变量，为此，只需在声明语句中的变量名后加上等号和初始值即可：

```
int count = 0;
double percent = 0.01, taxrate = 28.5;
```

第一条语句声明一个名为 count 的整型变量，并将其值初始化为 0；第二条语句声明并初始化两个 double 变量，其中第一个变量 percent 被初始化为 0.01，而第二个变量 taxrate 被初始化为 28.5。

请不要将变量初始化为允许范围之外的值，下面是两个这样的例子：

```
int weight = 100000;
unsigned int value = -2500;
```

C 编译器也许不会发现这种错误，因此程序将被编译和链接，但当程序运行时，结果将可能与期望的不同。

应　该	不 应 该
一定要了解变量占用的字节数。 应使用 typedef 提高程序的可读性。 声明变量时，应尽可能对它进行初始化。	不要使用未被初始化的变量，否则结果可能出乎意料。 存储型数据时，不要使用 float 或 double 变量，虽然这样做不会出现问题，但效率不高。 不要试图将超出变量取值范围的值赋给变量。 不要将负值赋给 unsigned 变量。

3.4 常　量

和变量一样，常量也是程序使用的一个数据存储位置；与变量不同的是，在程序运行期间，存储在常量中的值是不能修改的。C 语言中有两种常量，每种常量都有其特定的用途：

- 字面常量；
- 符号常量。

3.4.1 字面常量

字面常量是在源代码中直接输入的值，下面是两个字面常量的例子：

```
int count = 20;
float tax_rate = 0.28;
```

其中 20 和 0.8 是字面常量。上述两条语句将这些值分别存储到变量 count 和 tax_rate 中。注意，其中一个常量包含小数点，而另一个没有。是否有小数点决定了常量是整型常量还是浮点数常量。

包含小数点的字面常量是浮点数常量，C 编译器将其表示为一个双精度数字。可以使用标准的小数表示

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

法来书写浮点数常量，如下面的范例所示：

```
123.456
0.019
100.
```

注意，第三个常量 100.虽然是一个整数（没有小数部分），但书写时仍然包含小数点。小数点导致 C 编译器将该常量视为一个双精度值；如果没有小数点，则将被视为一个整型常量。

也可以使用科学计数法来书写浮点常量。高中数学中介绍过，科学计数法将数字表示为一个小数和 10 的正数或负数次幂之间的乘积。科学计数法在表示特别大或特别小的数字时很有用。在 C 语言中，科学计数法被表示为小数、E 或 e 以及指数：

- 1.23E2：1.23 乘以 10 的 2 次方，即 123；
- 4.08e6：4.08 乘以 10 的 6 次方，即 4080000；
- 0.85e-4：0.85 乘以 10 的-4 次方，即 0.000085。

不带小数点的常量将被编译器表示为一个整数。整型常量可以使用三种不同的表示方式来书写：

- 首位不为 0 的常量被视为 10 进制整数。十进制常量可以包含数字 0～9，并可以在最前面加上加号或减号（没有加号或减号时视为正数）。
- 以 0 打头的常量被视为八进制整数。八进制常量可以包含数字 0～7，并可以在前面加上加号或减号。
- 以 0x 或 0X 打头的常量被视为十六进制整数。十六进制常量可以包含数字 0～9 和字母 A～F，并可以在前面加上加号或减号。



注意：有关十进制和十六进制的详细信息，请参阅附录 C。

3.4.2　符号常量

符号常量是程序中用名称（符号）表示的常量。和字面常量一样，符号常量也不能修改。当您在程序中需要这种常量的值时，可以使用其名称，就像使用变量名一样。符号常量的实际值只需输入一次，这是在首次定义它时完成的。

与字面常量相比，符号常量有两个重要的优点，如下面的范例所示。假设您要编写一个执行各种几何计算的程序，则经常需要用到 π 的值（3.14）（几何课介绍过，π 是圆的周长和直径的比值）。例如，当半径已知时，要计算圆的周长和面积，可以使用下面的公式：

```
circumference = 3.14 * (2 * radius);
area = 3.14 * (radius)*(radius);
```

在 C 语言中，星号（*）表示乘号，这将在第 4 天的课程中介绍。因此，第一条语句的含义为，将变量 radius 的值乘以 2，再将结果乘以 3.14，然后将结果赋给变量 circumference。

然而，如果定义了一个名为 PI、值为 3.14 的符号常量，则可以这样编写代码：

```
circumference = PI * (2 * radius);
area = PI * (radius)*(radius);
```

上述代码更为清晰。您不用猜测值 3.14 有何用途，而可以直接看到常量 PI。

当您需要修改常量的值时，符号常量的第二个优点便显现出来了。还是以前面的范例为例，为提高程序的精度，您可能需要将 PI 定义为一个包含更多小数位的值（3.14159），而不是 3.14。如果您使用的是字面常量，则必须在源代码中找到每个 3.14，并将其修改为 3.14159；但如果使用的是符号常量，则只需在定义该常量的地方进行一次修改，而无需修改代码的其他地方。

1．定义符号常量

在 C 语言中，定义符号常量的方式有两种：使用编译指令#define 或使用关键字 const。编译指令#define 的用法如下：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
#define CONSTNAME literal
```

这将创建一个名为 CONSTNAME、值为 literal 的常量，其中 literal 是一个字面常量。CONSTNAME 遵循的规则和变量相同。根据约定，符号常量名中的字母为大写，这样易于将其同变量名区分开来。根据约定，变量名中的字母为小写。对于前面的范例而言，定义常量 PI 的#define 编译指令如下：

```
#define PI 3.14159
```

注意，#define 语句不以分号结尾。#define 可以位于源代码的任何位置，不过它定义的常量只在后面的源代码中有效。最常见的情况是，程序员将所有的#define 放在一起，并将它们放在 main()函数之前。

2. #define 的工作原理

#define 编译指令的准确含义是，命令编译器“将源代码中所有的 CONSTNAME 替换为 literal”。其效果与使用编辑器手工进行查找并替换相同。注意，#define 并不会将长名称中、双引号中和程序注释中的内容进行替换。例如，在下面的代码中，第 2 和 3 行的“PI”不会被替换。

```
#define PI 3.14159
/* You have defined a constant for PI. */
#define PIPETTE 100
```



注意：编译指令#define 是 C 语言中的预处理器编译指令之一，有关它的详细讨论，请参阅第 21 天的课程。

3. 使用关键字 const 来定义常量

第二种定义符号常量的方式是使用关键字 const。const 是一个修饰符，可用于任何变量声明中。被声明为 const 的变量在程序执行期间不能被修改，声明时被初始化为一个值，以后便不能修改。下面是一些例子：

```
const int count = 100;
const float pi = 3.14159;
const long debt = 12000000, float tax_rate = 0.21;
```

const 将影响声明行中的所有变量。在最后一行中，debt 和 tax_rate 都是符号常量。顺便说一句，这里 debt 的类型被声明为 long，而 tax_rate 则被声明为 float 类型。

如果程序试图修改 const 变量的值，编译器将生成一条错误消息。下面的代码将发生错误：

```
const int count = 100;
count = 200;       /* Does not compile! Cannot reassign or alter */
                   /* the value of a constant. */
```

使用编译指令#define 和关键字 const 创建的符号常量之间有什么实际区别呢？差别涉及到指针和变量作用域。指针和变量作用域是 C 语言编程中的两个重要方面，这将在第 9 天和第 12 天的课程中介绍。

下面的程序演示了如何声明变量以及使用字面常量和符号常量。程序清单 3.2 提示您输入您的体重和出生年份，然后计算并显示您的体重（单位为克）以及 2010 年时您的年龄。您可以按第 1 天课程中介绍的步骤输入、编译并运行该程序。



注意：今天的大多数程序在声明常量时都使用 const，而不是#define。

程序清单 3.2　　const.c：演示变量和常量的用法

```
1:   /* Demonstrates variables and constants */
2:   #include <stdio.h>
3:
4:   /* Define a constant to convert from pounds to grams */
5:   #define GRAMS_PER_POUND 454
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
6:
7:    /* Define a constant for the start of the next century */
8:    const int TARGET_YEAR = 2010;
9:
10:   /* Declare the needed variables */
11:   long weight_in_grams, weight_in_pounds;
12    int year_of_birth, age_in_2010;
13:
14:   int main( void )
15:   {
16:       /* Input data from user */
17:
18:       printf("Enter your weight in pounds: ");
19:       scanf("%d", &weight_in_pounds);
20:       printf("Enter your year of birth: ");
21:       scanf("%d", &year_of_birth);
22:
23:       /* Perform conversions */
24:
25:       weight_in_grams = weight_in_pounds * GRAMS_PER_POUND;
26:       age_in_2010 = TARGET_YEAR - year_of_birth;
27:
28:       /* Display results on the screen */
29:
30:       printf("\nYour weight in grams = %ld", weight_in_grams);
31:       printf("\nIn 2010 you will be %d years old\n", age_in_2010);
32:
33:       return 0;
34:   }
```

该程序的运行情况如下：

```
Enter your weight in pounds: 175
Enter your year of birth: 1965

Your weight in grams = 79450
In 2010 you will be 45 years old
```

分析：该程序在第 5 和 8 行声明了两种符号常量。第 5 行使用一个常量来使 454 更容易理解。由于第 25 行使用的是 GRAMS_PER_POUND，因此更容易理解。第 11 和 12 行声明了该程序使用的变量，这里使用了诸如 weight_in_grams 等描述性名称，这样您便可以知道该变量的用途。第 18 和 20 行在屏幕上打印提示语。后面将更详细的介绍 printf()函数。为让用户能够对提示做出响应，第 19 和 21 行使用了另一个库函数 scanf()（将在后面介绍）从屏幕获取信息。第 25 和 26 行计算用户的体重（单位为克）以及到 2010 年时用户的年龄。这些语句以及其他一些语句将在明天的课程中进行介绍。最后，第 30 和 31 行将结果显示给用户。

应　该	不 应 该
应使用常量来提高程序的可读性	初始化常量后，不要试图给它赋值。

3.5　总　结

今天的课程介绍了数值变量，C 程序使用它来存储数据。数值变量分为两大类：整型变量和浮点变量，

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

其中每一类又被分为多类。到底使用哪种变量（int、long、float 或 double），取决于该变量将被用来存储的数据的性质。另外，在 C 程序中，使用变量之前必须声明它。变量声明将变量的名称和类型告知编译器。

另外，您还学习了 C 语言中的两种常量：字面常量和符号常量。和变量不同，常量的值在程序执行期间是不能修改的。每当需要该值时，您便在源代码中输入字面常量；符号常量被指定了一个名称，每当需要该常量值时，便使用该名称。符号常量可以使用编译指令#define 或关键字 const 来创建。

3.6 问与答

问：long int 变量能够存储更大的数字，为何不总是使用这种变量，而不使用 int 变量？

答：long int 变量占用的内存比 int 变量多。在小型程序中，这不会引起问题。然而，当程序逐渐变大时，应尽可能高效地使用内存。

问：如果将一个小数赋给整型变量，将出现什么样的情况？

答：可以将一个小数赋给 int 变量，但小数部分将被截去。如果该变量是一个常量，则编译器可能发出警告。例如，将 3.14 赋给整型变量 pi 时，pi 的值将为 3.14 被截去并被丢弃。

问：如果将一个超出变量取值范围上限的数值赋给变量，情况将如何？

答：很多编译器允许这样做，不会产生错误。该数字将被回绕，以便能够存储到变量中，因此结果将是错误的。例如，将 32768 赋给一个两字节的有符号 short 变量时，该变量的将实际包含值为-32768；如果将 65535 赋给该变量时，实际值将为-1。通常，实际存储的值为赋给的值减去变量所占内存能够存储的最大值。

问：将负数赋给无符号变量将出现什么情况？

答：正如前面指出的，如果您这样做，编译器可能不会产生错误。但编译器就像您赋给的值过大那样，执行回绕操作。例如，将-1 赋给一个两字节的无符号 int 变量时，编译器将把可能的最大值（65535）存储到该变量中。

问：使用编译指令#define 和关键字 const 定义的符号常量之间的实际差别何在？

答：这种差别涉及到指针和变量作用域。指针和变量作用域是 C 语言编程中的两个重要方面，将在第 9 天和第 12 天的课程中介绍。

3.7 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

3.7.1 小测验

1. 整型变量和浮点变量之间有何差别？
2. 指出使用双精度浮点变量（double 类型）而不是单精度浮点变量（float 类型）的两个原因。
3. 在变量长度方面，有哪 5 条规则总是正确的？
4. 与使用字面常量相比，使用符号常量有哪两个优点？
5. 使用两种方法定义值为 100 的符号常量 MAXIMUM。
6. 在 C 语言中，变量名可以包含哪些字符？
7. 给变量和常量命名时，必须遵循哪些规则？
8. 符号常量和字面常量之间有何区别？
9. int 变量能够存储的最小值是多少？

3.7.2 练习

1. 对于下面的各种值，最适合使用何种类型的变量来存储？

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

a. 人的年龄；
b. 人的体重（单位为克）；
c. 圆的半径；
d. 您的年薪；
e. 商品的价格；
f. 考试的最高分（假设总是为 100）；
g. 温度；
h. 个人的净资产；
i. 到某个星球的距离（单位为英里）。

2. 对于练习 1 中的变量，取一个合适的名称。
3. 声明练习 2 中的各个变量。
4. 下面的哪些变量名是合法的？

a. `123variable`
b. `x`
c. `total_score`
d. `Weight_in_#s`
e. `one`
f. `gross-cost`
g. `RADIUS`
h. `Radius`
i. `radius`
j. `this_is_a_variable_to_hold_the_width_of_a_box`

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 4 天课程　语句、表达式和运算符

C 程序是由语句组成的，而大多数语句又是由表达式和运算符组成的。要编写 C 程序，必须理解语句、表达式和运算符。今天将介绍以下内容：

- 语句是什么？
- 表达式是什么？
- 如何使用 C 语言中的数学、关系和逻辑运算符？
- 什么是运算符优先级？
- if 语句。

4.1　语　句

语句是一条完整的指令，命令计算机执行特定的任务。在 C 语言中，通常每条语句占一行，虽然有些语句占多行。C 语句总是以分号结尾（诸如#define 和#include 等预处理器编译指令除外，这些编译指令将在第 21 天的课程中讨论）。前面已经介绍过一些类型的语句。例如：

```
x = 2 + 3;
```

是一条赋值语句，它命令计算机将 3 和 2 相加，并将结果赋给变量 x。

4.1.1　空白对语句的影响

空白指的是源代码中的空格、水平制表符、垂直制表符和空行。C 编译器忽略空白。当编译器读取源代码中的语句时，查找语句中的字符和末尾的分号，但忽略空白。因此语句：

```
x=2+3;
```

和下面的语句等价：

```
x = 2 + 3;
```

也同下面的语句等效：

```
x       =
2
     +
3  ;
```

这在格式化源代码时有很大的灵活性，但不应采用上面这样的格式。输入语句时，每条语句应占一行，并采用标准的模式，即在变量和运算符的两边加上空格。采用本书的格式化约定就不错。随着经验的增长，您可能喜欢采用稍微不同的约定。重要的是确保源代码易于阅读。

C 语言忽略空白这一规则存在一种例外情况。字面字符串常量中的制表符和空白不被忽略，它们被视为字符串的组成部分。字符串是一系列的字符；而字面字符串常量用引号括起，编译器逐字的解释它，而不忽略其中的空格。下面是一个字面字符串：

```
"How now brown cow"
```

该字面字符串不同于下面的字符串：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
"How  now  brown  cow"
```

差别在于后者包含更多的空格。对于字面字符串，编译器将记录其中的空白。

虽然下述代码的格式很糟糕，但却是合法的：

```
printf(
"Hello, world!"
);
```

但下面的代码不合法：

```
printf("Hello,
world!");
```

要将字面字符串常量放在多行中，必须在换行之前加上反斜杠（\）。因此下面的代码是合法的：

```
printf("Hello,\
world!");
```

4.1.2　创建空语句

让一个分号单独占一行，便创建了一条空语句。空语句不执行任何操作，但在 C 语言中是完全合法的。本书的后面将介绍空语句的用途。

4.1.3　使用复合语句

复合语句也叫代码块，是一组用花括号括起的语句。下面便是一个代码块：

```
{
        printf("Hello, ");
        printf("world!");
}
```

在 C 语言中，可以使用单条语句的地方便可以使用代码块。也可以以其他方式排列花括号，下面的代码块同前一个例子等价：

```
{printf("Hello, ");
printf("world!");}
```

让花括号单独占一行是个不错的主意，这样语句块的开始和结束位置便清晰明了，同时也容易发现遗漏了花括号的情况。

应　该	不 应 该
使用空白的方式应始终一致； 应让语句块中的花括号占一行，这样代码更易于阅读； 花括号应对齐，这样容易找到代码块的开始和结束位置。	在不必要的情况下，不应让单条语句占多行； 将字符串分多行书写时，别忘了在行尾加上反斜杠。

4.2　表达式

在 C 语言中，表达式可以是任何计算结果为数值的东西。C 语言中，有各种复杂程度不同的表达式。

4.2.1　简单表达式

最简单的表达式只包含一项：一个简单变量、字面常量或符号常量。下面是 4 个这样的表达式：

- PI：程序中定义的符号常量；
- 20：字面常量；
- rate：变量；
- -1.25：字面常量。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

字面常量的计算结果等于它本身的值；符号常量的计算结果等于使用#define 编译指令定义它时给它指定的值；变量的计算结果等于程序赋给它的当前值。

4.2.2 复杂表达式

复杂表达式由多个更简单的表达式组成，表达式之间用运算符连接。例如表达式：

```
2 + 8
```

由于表达式 2 和 8 以及加法运算符+组成，其计算结果为 10。您可以编写非常复杂的表达式：

```
1.25 / 8 + 5 * rate + rate * rate / cost
```

当表达式包含多个运算符时，如何计算表达式取决于运算符的优先级。有关运算符的优先级以及运算符的所有细节将在今天课程的后面介绍。

表达式还可以更有趣。请看下面的赋值语句：

```
x = a + 10;
```

上述语句计算表达式 a + 10 的值，并将结果赋给 x。另外，整条语句本身也是一个表达式，其结果为等号左边的变量的值。图 4.1 说明了这一点。

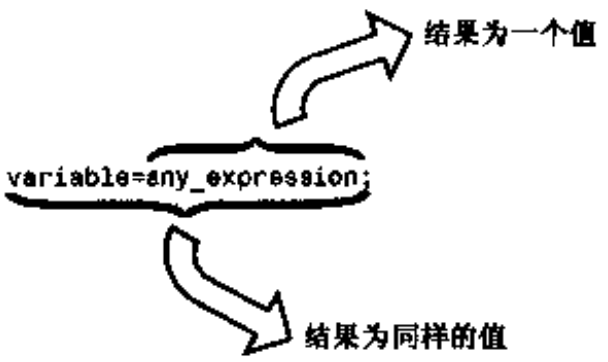


图 4.1 赋值语句本身也是一个表达式

因此可以编写下面这样的语句，该语句将表达式 a + 10 的值赋给变量 x 和 y：

```
y = x = a + 10;
```

也可以编写下面这样的语句：

```
x = 6 + (y = 4 + 5);
```

该语句的运行结果为：y 的值为 9，而 x 的值为 15。为使上述语句能够通过编译，其中的圆括号是必不可少的。有关圆括号的用法将在今天课程的后面介绍。



注意：除了本书将指出的一些情况外，不应将赋值语句嵌套到其他表达式中。

4.3 运算符

运算符是一个命令编译器对一个或多个操作数执行某种运算的符号。操作数是运算符对其执行操作的东西。在 C 语言中，所有的操作数都是表达式。运算符分为以下几类：

- 赋值运算符；
- 数学运算符；
- 关系运算符；
- 逻辑运算符。

4.3.1 赋值运算符

赋值运算符是一个等号（=），在编程中，其用途与数学中不同。在 C 程序中，下面的语句：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
x = y;
```

指的是将 y 的值赋给 x，而不是 x 等于 y。赋值语句的右边可以是任何表达式，但左边必须是一个变量名，因此其格式如下：

```
variable = expression;
```

当这种语句被执行时，将计算表达式的值，并将结果赋给变量。

4.3.2　数学运算符

C 语言中的数学运算符执行诸如加和减等数学运算。C 语言中有 2 种单目数学运算符和 5 种双目数学运算符。

1．单目数学运算符

单目数学运算符只需要一个操作数。C 语言中有两种单目数学运算符，如表 4.1 所示。

表 4.1　　单目数学运算符

运 算 符	符　号	操　作	范　例
递增	++	将操作数加 1	++x、x++
递减	--	将操作数减 1	--x、x--

递增和递减运算符只能用于变量，而不能用于常量，它们分别将操作数加 1 和减 1。换句话说，下列表达式：

```
++x;
--y;
```

和下面的表达式等价：

```
x = x + 1;
y = y - 1;
```

从表 4.1 可知，这两个运算符都可以放在操作数的前面（前缀模式），也可以放在操作数的后面（后缀模式）。这两种模式并不等效，它们之间的区别在于何时执行递增或递减操作：

- 采用前缀模式时，先执行递增或递减运算，再计算表达式的值。
- 采用后缀模式时，先计算表达式的值，然后再执行递增或递减操作。

我们通过一个例子来说明这一点。请看下面的两条语句：

```
x = 10;
y = x++;
```

这两条语句被执行后，x 的值为 11，而 y 的值为 10。首先将 x 的值赋给 y，然后将 x 的值加 1。而下面的语句被执行后，x 和 y 的值都为 11，即首先将 x 的值加 1，然后将 x 的值赋给 y：

```
x = 10;
y = ++x;
```

=是一个赋值运算符，而不是说两个量相等。可以将=看作是一个“复印”运算符。语句 y = x 指的是将 x 的值复制给 y。复制结束后，如果 x 的值发生变化，将不会影响 y。

程序清单 4.1 中的程序说明了前缀模式和后缀模式之间的区别。

程序清单 4.1　　unary.c：说明前缀模式和后缀模式之间的差别

```
1:  /* Demonstrates unary operator prefix and postfix modes */
2:
3:  #include <stdio.h>
4:
5:  int a, b;
6:
7:  int main( void )
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
8:  {
9:      /* Set a and b both equal to 5 */
10:
11:     a = b = 5;
12:
13:     /* Print them, decrementing each time. */
14:     /* Use prefix mode for b, postfix mode for a */
15:
16:     printf("\nPost  Pre");
17:     printf("\n%d    %d", a--, --b);
18:     printf("\n%d    %d", a--, --b);
19:     printf("\n%d    %d", a--, --b);
20:     printf("\n%d    %d", a--, --b);
21:     printf("\n%d    %d\n", a--, --b);
22:
23:     return 0;
24: }
```

该程序的输出如下：

```
Post  Pre
5     4
4     3
3     2
2     1
1     0
```

分析：该程序的第5行声明了两个变量a和b，第11行将这两个变量的值都设置为5。每执行第17～21行的一条printf()语句后，a和b的值都被减1。但对于变量a，是在被打印后再减1；而变量b是在打印前被减1。



注意：第2天的课程介绍了另一个单目运算符sizeof。您可能认为运算符应该是符号，但关键字sizeof实际上也是一个运算符。

2. 双目数学运算符

双目运算符接受两个操作数。表4.2列出了双目运算符，其中包括计算器上常用的数学运算。

表4.2　　双目数学运算符

运算符	符　号	操　作	范　例
加法	+	将两个操作数相加	x+y
减法	-	将第1个操作数减去第2个操作数	x-y
乘法	*	将两个操作数相乘	x*y
除法	/	将第1个操作数除以第2个操作数	x/y
求模	%	第1个操作数除以第2个操作数得到的余数	x%y

对于表4.2中的前4个运算符，读者应该很熟悉，使用起来也不会有任何问题。第5个运算符求模可能初次见到。求模返回第一个操作数除以第二个操作数得到的余数。例如，11%4的结果为3（即11＝4×2＋3）。下面是一些其他的例子：

```
100 modulus 9 equals 1
10 modulus 5 equals 0
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
40 modulus 6 equals 4
```

程序清单 4.2 演示了如何使用求模运算符将秒数转换为小时、分钟和秒。

程序清单 4.2　　seconds.c：演示如何使用求模运算符

```
1:  /* Illustrates the modulus operator. */
2:  /* Inputs a number of seconds, and converts to hours, */
3:  /* minutes, and seconds. */
4:
5:  #include <stdio.h>
6:
7:  /* Define constants */
8:
9:  #define SECS_PER_MIN 60
10: #define SECS_PER_HOUR 3600
11:
12: unsigned seconds, minutes, hours, secs_left, mins_left;
13:
14: int main( void )
15: {
16:     /* Input the number of seconds */
17:
18:     printf("Enter number of seconds (< 65000): ");
19:     scanf("%d", &seconds);
20:
21:     hours = seconds / SECS_PER_HOUR;
22:     minutes = seconds / SECS_PER_MIN;
23:     mins_left = minutes % SECS_PER_MIN;
24:     secs_left = seconds % SECS_PER_MIN;
25:
26:     printf("%u seconds is equal to ", seconds);
27:     printf("%u h, %u m, and %u s\n", hours, mins_left, secs_left);
28:
29:     return 0;
30: }
```

该程序的运行情况如下：

```
Enter number of seconds (< 65000): 60
60 seconds is equal to 0 h, 1 m, and 0 s
Enter number of seconds (< 65000): 10000
10000 seconds is equal to 2 h, 46 m, and 40 s
```

分析：程序 seconds.c 采用的格式与前面的程序相同。第 1～3 行是注释，指出了该程序的功能。第 4 行是空白，旨在提高程序的可读性。同语句和表达式中的空白一样，空白行也会被编译器忽略。第 5 行包含了程序所需的头文件。第 9 和 10 行定义了两个常量：SECS_PER_MIN 和 SECS_PER_HOUR，用于提高程序中语句的可读性。第 12 行声明了要使用的所有变量。有些人选择在每一行中声明一个变量，而不是在一行中声明所有的变量。和 C 语言中的很多内容一样，这也只是一个风格方面的问题。这两种方式都是正确的。

第 14 行的 main()函数是程序的主体部分。为将秒数转换为小时数和分钟数，程序必须首先获得秒数。为此，第 18 行使用 printf()函数提示用户输入秒数，然后第 19 行使用 scanf()函数读取用户输入的数字，并将其存储到变量 seconds 中。有关函数 printf()和 scanf()的更详细的信息，请参阅第 7 天的课程。第 21 行的表达式将秒数除以常量 SECS_PER_HOUR 来计算小时数。由于 hour 是一个整型变量，因此余数被忽略。第 22 行采用同样的逻辑来计算总分钟数。由于第 22 行计算得到的总分钟数包含小时中的分钟数，因此第 23 行

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

使用求模运算符进行剔除，并获得余下的分钟数。第 24 行执行类似的计算，以确定余下的秒数。第 26 和 27 行同您前面见到的语句类似，它们接受计算得到的值，并显示它们。第 29 行在退出之前将 0 返回给操作系统，从而结束程序。

4.3.3 运算符优先级和圆括号

在包含多个运算符的表达式中，运算的执行顺序是什么样的呢？下面的赋值语句说明了这一问题的重要性：

```
x = 4 + 5 * 3;
```

如果先执行加法运算，结果将如下，因此 x 的值将为 27：

```
x 9 * 3;
```

相反，如果先执行乘法运算，结果将如下，因此 x 的值将为 19：

```
x = 4 + 15;
```

显然，必须制定一些有关运算顺序的规则。这种顺序被称为运算符优先级，C 语言对此有严格的规定。每个运算符都有一个优先级。计算表达式时，首先执行优先级高的运算符。表 4.3 列出了数学运算符的优先级。1 表示优先级最高，因此首先计算。

表 4.3　　数学运算符的优先级

运 算 符	相对优先级
++、--	1
*、/、%	2
+、-	3

从表 4.3 可知，表达式中各种运算的执行顺序如下：

- 单目递增和递减；
- 乘法、除法和求模；
- 加法和减法。

如果表达式中包含多个优先级相同的运算符，则按从左到右的顺序计算。例如，在下面的表达式中，%和*的优先级相同，但%位于最左边，因此首先被执行：

```
12 % 5 * 2;
```

上述表达式的结果为 4（12 % 5 的结果为 2，2 乘以 2 等于 4）。

回到前面的语句 x = 4 + 5 * 3，它将 19 赋给 x，因为先执行乘法运算，再执行加法运算。

如果表达式不能按您期望的那样运算，该如何办呢？还是以上述范例为例，如果您要将 4 和 5 相加，然后将结果乘以 3，该如何办呢？C 语言使用圆括号来改变计算顺序。位于圆括号内的子表达式将首先被计算，而不考虑优先级。因此，您可以这样编写：

```
x = (4 + 5) * 3;
```

位于圆括号内的表达式 4 + 5 首先被计算，因此赋给 x 的值为 27。

在表达式中，可以使用多个圆括号，并且可以嵌套。当圆括号被嵌套时，从内向外计算表达式。请看下述复杂的表达式：

```
x = 25 - (2 * (10 + (8 / 2)));
```

该表达式的计算顺序如下：

1. 首先计算最内面的表达式 8 / 2，结果为 4，因此整个表达式变为：

```
25 - (2 * (10 + 4))
```

2. 接着计算表达式 10 + 4，结果为 14，因此整个表达式变为：

```
25 - (2 * 14)
```

3. 计算最外面的表达式 2 * 14，结果为 28，因此整个表达式变为：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
25 -28
```

4. 最后计算表达式 25－28，并将结果-3 赋给变量 x：

```
x = -3
```

在有些表达式中，为清晰起见，即使不会改变运算符优先级，您也可能想添加圆括号。圆括号必须成对出现，否则编译器将产生错误消息。

4.3.4　子表达式的计算顺序

正如前一节指出的，如果表达式包含多个优先级相同的表达式，它们将从左到右依次计算。例如，在下面的表达式中：

```
w * x / y * z
```

首先将 w 和 x 相乘，然后将得到的结果除以 y，再乘以 z。

但如果中间有其他优先级的运算符，则不能保证依次从左到右进行计算。请看下面的表达式：

```
w * x / y + z / y
```

由于优先级的缘故，将先计算乘除，后计算加法。然而，C 语言并没有规定是先计算子表达式 w * x / y 还是 z / y。就这个表达式而言，这可能毫无关系。请看下面的表达式：

```
w * x / ++y + z / y
```

如果先计算左边的子表达式，则计算第二个表达式时，y 的值已经加 1；如果先计算右边的表达式，则 y 的值没有加 1，这样，结果将不同。因此，在编程中，应避免使用这种不确定的表达式。

在今天课程的最后一节“再谈运算符优先级”列出了 C 语言中所有运算符的优先级。

应　该	不 应 该
应使用圆括号使表达式的运算顺序清晰明了。	表达式不应过长。将表达式分成两条或更多的语句，通常会更为清晰，尤其是使用单目运算符（++和--）时。

4.3.5　关系运算符

关系运算符用于比较表达式，提出诸如“x 是否大于 100”或“y 是否等于 0”等问题。含有关系运算符的表达式的结果为 true（1）或 false（0）。表 4.4 列出了 C 语言中的 6 个关系运算符。

表 4.5 列出了一些如何使用关系运算符的范例，这些范例使用的是字面常量，但其中的原理也适用于变量。



注意：“true”与“yes”和 1 的含义相同；而“false”与“no”和 0 的含义相同。

表 4.4　C 语言中的关系运算符

运 算 符	符　号	提出的问题	范　例
等于	==	第一个操作数是否等于第二个操作数？	x==y
大于	>	第一个操作数是否大于第二个操作数？	x>y
小于	<	第一个操作数是否小于第二个操作数？	x<y
大于等于	>=	第一个操作数是否大于或等于第二个操作数？	x>=y
小于等于	<=	第一个操作数是否小于或等于第二个操作数？	x<=y
不等于	!=	第一个操作数和第二个操作数是否不相等？	x!=y

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

表 4.5　关系运算符的使用范例

表达式	含义	结果
5 == 1	5 等于 1 吗?	0（false）
5 > 1	5 大于 1 吗?	1（true）
5 != 1	5 不等于 1 吗?	1（true）
(5 + 10) == (3 * 5)	(5 + 10)等于(3 * 5)吗?	1（true）

应读	不应读
应了解 C 语言如何解释 true 和 false。使用关系运算符时，true 相当于 1，false 相当于 0。	不要将关系运算符==和赋值运算符=混淆，这是 C 语言程序员最常犯的错误之一。

4.4　if 语句

关系运算符主要用于构建 if 语句和 while 语句（将在第 6 天的课程中介绍）中的关系表达式。现在，介绍 if 语句的基本知识，说明如何使用关系运算符来构建程序控制语句。

您可能会问，程序控制语句是什么？C 程序中的语句通常是按其在源代码文件中的出现顺序从前到后依次执行的。程序控制语句用于改变语句的执行顺序，可能导致其他语句执行多次或更本不执行，这取决于条件。if 语句是 C 语言中的程序控制语句之一，其他程序控制语句（如 while 语句）将在第 6 天的课程中介绍。

if 语句的基本格式是，对一个表达式进行计算，根据计算结果决定是否执行后面的语句。if 语句的格式如下：

```
if (expression)
{
        statement;
}
```

如果 expression 为 true，则执行 statement；否则不执行。无论结果如何，接着都将执行 if 语句后面的语句。可以这么说，是否执行 statement 取决于 expression 的结果。if(expression)和 statement;一起组成了完整的 if 语句，它们并非两条独立的语句。

通过使用复合语句（代码块），if 语句能够控制是否执行多条语句。正如本章前面定义的，代码块是一组用花括号括起的语句。任何可使用单条语句的地方，都可以使用语句块。因此，可以像下面这样编写 if 语句：

```
if (expression)
{
        statement1;
        statement2;
        /* additional code gces here */
        statementn;
}
```

应读	不应读
请切记，一天编写过多的程序会得 C 语言病。 在代码块中，应对语句采取缩进格式，使其易于阅读，这包括 if 语句中的代码块中的语句。	



警告：不要在 if 语句表达式的后面加上分号，这是错误的。if 语句应以条件语句结束。在下面的代码中，由于其中的分号，不管 x 是否等于 2，statement1 都将执行。分号导致其中的每一行都被视为一条独立的语句，而不是整个被视为一条语句：

```
if( x == 2);          /* semicolon does not belong!  */
statement1;
```

对于这种错误，编译器通常不会生成错误消息。

当您编程时将发现，if 语句最常与关系表达式一起使用，即“仅当条件为真时，执行下面的语句”。下面

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

是一个这样的例子：

```
if (x > y)
    y = x;
```

仅当 x 大于 y 时，上述代码才将 x 的值赋给 y；如果 x 不大于 y，则不执行赋值操作。程序清单 4.3 演示了 if 语句的用法。

程序清单 4.3　　list0403.c：演示 if 语句的用法

```
1:  /* Demonstrates the use of if statements */
2:
3:  #include <stdio.h>
4:
5:  int x, y;
6:
7:  int main( void )
8:  {
9:     /* Input the two values to be tested */
10:
11:    printf("\nInput an integer value for x: ");
12:    scanf("%d", &x);
13:    printf("\nInput an integer value for y: ");
14:    scanf("%d", &y);
15:
16:    /* Test values and print result */
17:
18:    if (x == y)
19:       printf("x is equal to y\n");
20:
21:    if (x > y)
22:       printf("x is greater than y\n");
23:
24:    if (x < y)
25:       printf("x is smaller than y\n");
26:
27:    return 0;
28: }
```

该程序的运行情况如下：

```
Input an integer value for x: 100

Input an integer value for y: 10
x is greater than y
Input an integer value for x: 10

Input an integer value for y: 100
x is smaller than y
Input an integer value for x: 10

Input an integer value for y: 10
x is equal to y
```

分析：list0403.c 使用了三个 if 语句（第 18～25 行）。您应熟悉该程序中的很多代码行。第 5 行声明了两个变量：x 和 y，第 11～14 行提示用户输入这些变量的值。第 18～25 行使用 if 语句判断 x 是大于、小于还

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

是等于 y。第 18 行使用一个 if 语句判断 x 是否等于 y。等于运算符==指的是“是否相等”，请不要将其同赋值运算符混淆。检查两个变量是否相等后，第 21 行检查 x 是否大于 y，然后第 24 行检查 x 是否小于 y。您也许认为这样做的效率不高，确实是这样。下一个程序将演示如何避免这种低效率。现在，运行该程序，分别给 x 和 y 指定不同的值，并查看运行结果。



注意：您可能注意到了，if 语句中的语句采取了缩进格式，这是一种常用的做法，旨在提高程序的可读性。

4.4.1 else 子句

if 语句中也可以包含 else 子句，方法如下：

```
if (expression)
        statement1;
else
        statement2;
```

如果 expression 为真，则执行 statement1 语句；否则执行 statement2 语句。statement1 和 statement2 都可以是复合语句（代码块）。

程序清单 4.4 使用包含 else 子句的 if 语句重新编写了程序清单 4.3 中的程序。

程序清单 4.4　　list0404.c：使用包含 else 子句的 if 语句

```
1:  /* Demonstrates the use of if statement with else clause */
2:
3:  #include <stdio.h>
4:
5:  int x, y;
6:
7:  int main( void )
8:  {
9:      /* Input the two values to be tested */
10:
11:     printf("\nInput an integer value for x: ");
12:     scanf("%d", &x);
13:     printf("\nInput an integer value for y: ");
14:     scanf("%d", &y);
15:
16:     /* Test values and print result */
17:
18:     if (x == y)
19:         printf("x is equal to y\n");
20:     else
21:         if (x > y)
22:             printf("x is greater than y\n");
23:         else
24:             printf("x is smaller than y\n");
25:
26:     return 0;
27: }
```

该程序的运行情况如下：

```
Input an integer value for x: 99
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
Input an integer value for y: 8
x is greater than y
Input an integer value for x: 8

Input an integer value for y: 99

x is smaller than y
Input an integer value for x: 99
Input an integer value for y: 99
x is equal to y
```

分析：第 18～24 行同前一个程序清单稍有不同。第 18 行仍然是判断 x 是否等于 y，如果相等，则像程序清单 4.3 一样，在屏幕上打印 x is equal to y。但然后程序将结束，第 20～24 行不会被执行。如果 x 不等于 y（或者更准确地说，如果表达式 x ==y 为假），将执行第 21 行。第 21 行判断 x 是否大于 y。如果是，则执行第 22 行——打印 x is greater than y；否则执行第 24 行。

程序清单 4.4 使用了一个嵌套的 if 语句。嵌套指的是将一个或多个 C 语句放置在另一条 C 语句中。在程序清单 4.4 中，第二条 if 语句是第一条 if 语句的 else 子句的一部分。

If 语句的语法如下：

格式 1：

```
if( expression )
{
        statement1;
}
next_statement;
```

这是最简单的 if 语句。如果 expression 为真，则执行 statement，否则不执行。

格式 2：

```
if( expression )
{
        statement1;
}
else
{
        statement2;
}
next_statement;
```

这种格式的 if 语句最常用。如果 expression 为真，则执行 statement1；否则执行 statement2。

格式 3：

```
if( expression1)
        statement1;
else if( expression2 )
        statement2;
else
        statement3;
next_statement;
```

这是嵌套 if 语句。如果第一个表达式 expression1 为真，则执行 statement1，然后执行 next_statement；否则，判断第二个表达式 expression2。如果该表达式为真，则执行 statement2 语句；否则执行 statement3。在这三条语句中，将只有一条被执行。

范例 1：

```
if( salary > 45,000 )
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
{
        tax = .30;
}
else
{
        tax = .25;
}
```

范例2：

```
if( age < 18 )
        printf("Minor");
else if( age < 65 )
        printf("Adult");
else
        printf( "Senior Citizen");
```

4.5 判断关系表达式

根据定义，使用关系运算符的表达式的结果为一个值。关系表达式的结果要么为假（0），要么为真（1）。虽然关系表达式最常见的用途是用于if语句和其他条件结构中，但也可被用作纯粹的数值，程序清单4.5演示了这一点。

程序清单4.5　　list0405.c：计算关系表达式

```
1:  /* Demonstrates the evaluation of relational expressions */
2:
3:  #include <stdic.h>
4:
5:  int a;
6:
7:  int main()
8:  {
9:      a = (5 == 5);          /* Evaluates to 1 */
10:     printf("\na = (5 == 5)\na = %d", a);
11:
12:     a = (5 != 5);          /* Evaluates to 0 */
13:     printf("\na = (5 != 5)\na = %d", a);
14:
15:     a = (12 == 12) + (5 != 1); /* Evaluates to 1 + 1 */
16:     printf("\na = (12 == 12) + (5 != 1)\na = %d\n", a);
17:     return 0;
18: }
```

该程序的输出如下：

```
a = (5 == 5)
a = 1
a = (5 != 5)
a = 0
a = (12 == 12) + (5 != 1)
a = 2
```

分析：乍一看，该程序清单的输出令人迷惑。使用关系运算符时，人们最常犯的错误是使用单个等号（赋

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

值运算符），而不是两个等号。下述表达式的结果为 5（同时将 5 赋给变量 x）：

```
x = 5;
```

而下面的表达式的结果为 0 或 1（取决于 x 是否等于 5），而且不会修改 x 的值：

```
x == 5
```

如果将它错误地写成：

```
if (x = 5)
      printf("x is equal to 5");
```

则总是会打印该消息，因此该 if 语句检测的表达式的结果总是为真，而不管 x 原来的值是多少。

看看程序清单 4.5，您将知道为何 a 会有那样的值。在第 9 行中，5 确实等于 5，因此将 true（1）赋给 a。在第 12 行，“5 不等于 5”为假，因此将 0 赋给 a。

这里重申，关系运算符用于创建关系表达式，询问两个表达式之间的关系。关系表达式返回的结果是一个数值：0（表示假）或 1（表示真）。

4.5.1　关系运算符的优先级

和数学运算符一样，每个关系运算符也都有优先级，当表达式中包含多个运算符时，这种优先级决定了以什么样的顺序执行这些运算符。同样，在使用关系运算符的表达式中，也可以使用圆括号来改变优先级。今天课程的最后一节“再谈运算符优先级”列出了所有运算符的优先级。

首先，所有关系运算符的优先级都低于数学运算符。因此，在下面的语句中，首先将 x 和 2 相加，然后将结果同 y 进行比较：

```
if (x + 2 > y)
```

上述语句同下面的语句等价。后者是一个使用圆括号使代码清晰明了的典范。

```
if ((x + 2) > y)
```

使用圆括号将 x + 2 括起，就是要将 x 与 2 的和同 y 进行比较，虽然编译器不要求这样做。

关系运算符的优先级有两种，如表 4.6 所示。

表 4.6　关系运算符的优先级

运 算 符	相对优先级
<、<=、>、>=	1
!=、==	2

因此下面的代码：

```
x == y > z
```

与下面的代码等效：

```
x == (y > z)
```

因为将首先计算表达式 y > z，结果为 0 或 1，然后判断 x 是否等于前一步的计算结果（0 或 1）。您很少会使用这样的结构，但您应该了解它们。

应　该	不 应 该
	不要在 if 语句的表达式中使用赋值语句，这可能让其他阅读代码的人感到迷惑。他们可能认为这是一个错误，而将赋值改为逻辑相等。 不要在包含 else 的 if 语句中使用“不等于”运算符（!=），使用等于运算符（==）总是更为清晰。例如，对于下面的代码： `if ( x != 5 )` `    statement1;` `else` `    statement2;` 最好修改为： `if (x == 5 )` `    statement2;` `else` `    statement1;`

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

4.6 逻辑运算符

有时候，您需要一次询问多个问题。例如，如果是工作日的早上 7 点，且我不是在休假，则提醒我。C 语言中的逻辑运算符让您能够将两个或更多的关系表达式组成一个结果为真或假的表达式。表 4.7 列出了 C 语言中的三种逻辑运算符。

表 4.7　　C 语言中的三种逻辑运算符

运算符	符号	范例
AND	&&	*exp1 && exp2*
OR	\|\|	*exp1 \|\| exp2*
NOT	!	*!exp1*

表 4.8 解释了这些逻辑运算符的工作原理。

表 4.8　　逻辑运算符的用法

表 达 式	结　果
(exp1 && exp2)	仅当 exp1 和 exp2 皆为真时为真（1），否则为假（0）。
(exp1 \|\| exp2)	仅当 exp1 和 exp2 皆为假时为假（0），否则为真（1）。
(!exp1)	如果 exp1 为真（1），则为假（0），否则为真。

从中可以知道，使用逻辑运算符的表达式的结果为真还是假，取决于其操作数的真/假值。表 4.9 列出了一些代码范例。

表 4.9　　使用逻辑运算符的代码范例

表 达 式	结　果
(5 == 5) && (6 != 2)	真（1），因为两个操作数都为真。
(5 > 1) \|\| (6 < 1)	真（1），因为有一个操作数为真。
(2 == 1) && (5 == 5)	假（0），因为有一个操作数为假。
!(5 == 4)	真（1），因为操作数为假。

可以创建包含多个逻辑运算符的表达式。例如，要询问“x 是等于 2、3 还是 4”，可以编写下面的表达式：

```
(x == 2) || (x == 3) || (x == 4)
```

逻辑运算符提供了多种询问问题的方式。如果 x 是一个整型变量，则前面的问题也可以以下面两种方式之一来提出：

```
(x > 1) && (x < 5)
(x >= 2) && (x <= 4)
```

4.7 再谈 true/false 值

前面介绍过，关系表达式的结果为 0（表示 false）或 1（表示 true）。然而，输入被用于表达式或期望逻辑值的语句中时，将被解释为 true 或 false，知道这一点很重要。使用的规则如下：

- 0 表示 false；
- 非零表示 true。

下面的范例说明了这一点，该范例打印 x 的值：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
x = 125;
if (x)
      printf("%d", x);;
```

由于 x 不为零，因此 if 语句认为表达式 x 的值为 true。还可以更普遍地利用这一规则：(epresssion)和下面的表达式等价：

```
(expression != 0)
```

如果 expression 的值不为零，则上述两个表达式都为 true，否则都为 false。使用非操作符（!），也可以这样编写：

```
(!expression)
```

上述代码与下面的代码等价：

```
(expression == 0)
```

4.7.1　运算符的优先级

您可能猜到了，逻辑运算符也有优先次序——无论是它们之间，还是相对于其他运算符。!运算符的优先级与单目数学运算符（++和--）相同，因此其优先级高于所有的关系运算符和所有的双目数学运算符。

而运算符&&和||的优先级要低得多——低于所有的数学运算符和关系运算符，虽然&&运算符的优先级高于||。和 C 语言中所有的运算符一样，使用逻辑运算符时，也可以使用圆括号来改变计算顺序。请看下面的范例：

您想编写一个逻辑表达式来完成下面三种比较：

1. a 是否小于 b?
2. a 是否小于 c?
3. c 是否小于 d?

您希望如果条件 3 为真，且条件 2 和条件 1 中的一个为真，则整个表达式为真。您可能这样编写代码：

```
a < b || a < c && c < d
```

然而，上述代码与您的初衷并不相符，因为运算符&&的优先级高于||。它等价于：

```
a < b || (a < c && c < d)
```

因此，如果 a<b，则无论关系 a<c 和 c<d 是否为真，结果都为真。因此，您应这样编写代码：

```
(a < b || a < c) && c < d
```

从而先执行运算符||，再执行&&。程序清单 4.6 对此进行了演示，它使用了这两种方式编写的表达式，同时对变量的值做了这样的设置，即在书写正确的情况下，表达式的结果为 false(0)。

程序清单 4.6　　list0406.c：逻辑运算符的优先级

```
1:  #include <stdio.h>
2:
3:  /* Initialize variables. Note that c is not less than d, */
4:  /* which is one of the conditions to test for. */
5:  /* Therefore, the entire expression should evaluate as false.*/
6:
7:  int a = 5, b = 6, c = 5, d = 1;
8:  int x;
9:
10: int main( void )
11: {
12:     /* Evaluate the expression without parentheses */
13:
14:     x = a < b || a < c && c < d;
15:     printf("\nWithout parentheses the expression evaluates as %d", x);
16:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
17:     /* Evaluate the expression with parentheses */
18:
19:     x = (a < b || a < c) && c < d;
20:     printf("\nWith parentheses the expression evaluates as %d\n", x);
21:     return 0;
22: }
```

该程序清单的输出如下：

```
Without parentheses the expression evaluates as 1
With parentheses the expression evaluates as 0
```

分析：请输入并运行该程序清单。请注意，打印的表达式的值是不同的。该程序的第 7 行将 4 个变量初始化为用于比较的值。第 8 行声明了用于存储和打印结果的变量 x。第 14 和 19 行使用了逻辑运算符。第 14 行没有使用圆括号，因此结果由优先级决定，但并不是您期望的。第 19 行使用圆括号改变表达式的计算顺序。

4.7.2 复合赋值运算符

C 语言的复合赋值运算符将双目数学运算符和赋值运算符组合在一起。例如，假设您要将 x 的值加 5，换句话说，将 x 和 5 相加，并将结果赋给 x，可以这样编写代码：

```
x = x + 5;
```

使用复合赋值运算符（可以将其看作是一种简明的赋值方式），可以这样编写代码：

```
x +=5;
```

复合赋值运算符的通用语法如下（其中 op 为双目运算符）：

```
exp1 op= exp2;
```

这与下面的书写方式等效：

```
exp1 = exp1 op exp2;
```

您可以使用本章前面介绍的 5 个双目数学运算符创建复合赋值运算符。表 4.10 列出了一些范例。

表 4.10　　复合赋值运算符范例

使用复合赋值运算符	相 当 于
x *= y	x = x * y
y -= z + 1	y = y - z + 1
a /= b	a = a / b
x += y / 8	x = x + y / 8
y %= 3	y = y % 3

复合赋值运算符提供了一种简单的方式，当赋值语句左边的变量名非常长时，其优势将特别明显。和其他所有赋值语句一样，复合赋值语句也是一个表达式，其值等于赋给左边变量的值。因此，执行下面的语句后，x 和 z 的值都将是 14：

```
x = 12;
z = x += 2;
```

4.7.3 条件运算符

条件运算符是 C 语言中唯一一个三目运算符（即接受三个操作数），其语法如下：

```
exp1 ? exp2 : exp3;
```

如果 exp1 为真（即非零），则整个表达式的结果为 exp2 的值；否则为 exp3 的值。例如，如果 y 为 true，则下面的语句将 1 赋给 x；否则将 100 赋给 x：

```
x = y ? 1 : 100;
```

同样，要将 x 和 y 中较大的一个值赋给 z，可以这样书写代码：

```
z = (x > y) ? x : y;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

也许您已经注意到了，条件运算符的功能有些类似于 if 语句。上述语句也可以编写成这样：

```
if (x > y)
z = x;
else
z = y;
```

并非所有能够使用 if...else 结构的情况都能使用条件运算符，但后者更简明。条件运算符也能用于不能使用 if 语句的地方，如调用另一个函数（如 printf()）：

```
printf( "The larger value is %d", ((x > y) ? x : y) );
```

4.7.4　逗号运算符

在 C 语言中，逗号常用作一个简单的标点符号，将变量声明、函数参数等分开。在某些情况下，逗号将是运算符，而不是分隔符。可以使用逗号将两个子表达式组合成一个表达式。结果如下：

- 两个表达式都将被计算，且先计算左边的表达式；
- 这个表达式的结果为右边的子表达式的值。

例如，下面的语句将 b 的值赋给 x，然后将 a 和 b 的值分别加 1：

```
x = (a++ , b++);
```

由于使用++运算符时，采用的是后缀模式，因此先将 b 的值赋给 x，然后再加 1。其中的圆括号是必不可少的，因为逗号运算符的优先级低于赋值运算符。

在明天的课程中您将知道，逗号运算符最常用于 for 语句中。

应　该	不应该		
应使用逻辑运算符&&和		，而不是嵌套 if 语句。	不要将赋值运算符（=）和等于运算符（==）混淆。

4.8　再谈运算符优先级

表 4.11 以优先级由高到低的方式列出了 C 语言中的运算符，位于同一行中的运算符的优先级相同。

表 4.11　　运算符的优先级

优先级	运算符
1	()[]->.
2	!~++--*（间接运算符）　&（地址运算符）
	sizeof +(单目) -(单目)
3	*(乘法运算符) / %
4	+-
5	<< >>
6	< <= > >=
7	== !=
8	& (按位 AND)
9	^
10	\|
11	&&
12	\|\|
13	?:
14	= += -= *= /= %= &= ^= \|= <<= >>=
15	,

()是函数运算符；[]是数组运算符

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com



提示：该表可供您参考，直到熟悉运算符的优先级为止。以后还需要参考该表。

4.9 总 结

今天介绍的内容很多。您知道了语句是什么，空白对 C 编译器而言是无关紧要的，语句总是以分号结尾。您还知道了，复合语句（代码块）是用花括号括起的多条语句，可用于能够使用单条语句的任何地方。

很多语句是由表达式和运算符组成的。表达式可以是结果为数值的任何东西。复杂的表达式可以包含多个更简单的表达式，后者也叫子表达式。

运算符是 C 语言中的符号，它命令计算机对一个或多个表达式执行某种运算。有些运算符是单目的，即对一个操作符进行运算；但大部分运算符是双目的，对两个操作数执行某种操作。有一个运算符——条件运算符是三目的。C 语言给运算符指定了不同的优先级，在包含多个运算符的表达式中，优先级决定了运算的执行顺序。

今天介绍的运算符分三类：

- 数学运算符：对操作数执行数学运算（如相加）；
- 关系运算符：对其操作数进行比较（如大于）；
- 逻辑运算符：对 true/false 表达式进行运算。C 语言分别使用 0 和 1 来表示 false 和 true，任何非零值都被视为 true。

今天的课程还介绍了 if 语句，它让您能够根据关系表达式的结果来控制程序的执行。

4.10 问与答

问：空格和空白行对程序的运行有何影响？

答：空白（空行、空格和制表符）可提高代码的可读性。当程序被编译时，空白将被删除，因此对可执行程序没有任何影响。因此，您应使用空白来提高程序的可读性。

问：使用复合 if 语句好，还是嵌套多个 if 语句好？

答：您应使代码易于理解。嵌套 if 语句时，表达式将按本章介绍的那样被判断。使用单条复合语句时，仅当整条语句为 false 时，表达式才会被判断。

问：单目运算符和双目运算符之间有何区别？

答：顾名思义，单目运算符使用一个变量，而双目运算符使用两个。

问：减法运算符（-）是单目的还是双目的？

答：既是单目的，也是双目的！编译器很聪明，能够判断出您使用的是哪种。它是通过表达式中的变量数目来判断这一点的。

在下面的语句中是单目的：

```
x = -y;
```

而在下面的语句中，则是双目的：

```
x = a - b;
```

问：负数被视为 true 还是 false？

答：请记住，0 被视为 false，其他任何值都被视为 true，包括负数。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

4.11　作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

4.11.1　小测验

1. 下面的语句是什么语句？含义是什么？

```
x = 5 + 8;
```

2. 表达式是什么？
3. 在包含多个运算符的表达式中，运算的执行顺序是由什么决定的？
4. 如果变量 x 的值为 10，下述两条语句分别执行后，x 和 a 的值分别是多少？

```
a = x++;
a = ++x;
```

5. 表达式 10 % 3 的值为多少？
6. 表达式 5 + 3 * 8 / 2 + 2 的值为多少？
7. 请重新书写问题 6 中的表达式，在其中添加圆括号，使其值为 16。
8. 如果表达式为假，则其值为多少？
9. 下述各对运算符中，哪一个的优先级更高？
 a. ==和<
 b. * 和+
 c. !=和==
 d. >=和>
10. 复合赋值运算符是什么？在什么情况下，它们很有用？

4.11.2　练习

1. 下述代码的格式不妥，请输入并编译它们，看其是否能够运行：

```
#include <stdio.h>
int x,y;int main(){ printf(
"\nEnter two numbers");scanf(
"%d %d",&x,&y);printf(
"\n\n%d is bigger",(x>y)?x:y);return 0;}
```

2. 重新编写练习 1 中的代码，提高其可读性。
3. 修改程序清单 4.1，使之向上数，而不是向下数。
4. 编写一个这样的 if 语句，即仅当 x 位于 1 到 20 之间时，将 x 的值赋给变量 y；如果 x 不在这个范围内，则保持 y 的值不变。
5. 使用条件运算符完成练习 4 中的任务。
6. 使用单条 if 语句和逻辑运算符重写下面的嵌套 if 语句：

```
if (x < 1)
   if ( x > 10 )
      statement;
```

7. 下述各个表达式的值分别是多少？
 a. (1 + 2 * 3)
 b. 10 % 3 * 3 - (1 + 2)
 c. ((1 + 2) * 3)
 d. (5 == 5)

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

e. (x = 5)

8. 如果 x = 4、y = 6，而 z = 2，则下述各个表达式分别是 true 还是 false。

a. if(x == 4)

b. if(x != y - z)

c. if(z = 1)

d. if(y)

9. 编写一条这样的 if 语句，即判断某个人是否是成年人（年龄大于 21），且不是老年人（年龄大于 65）。

10. 排错：修复下面的程序，使之能够正确地运行：

```
/* a program with problems... */
#include <stdio.h>
int x= 1;
int main( void )
{
    if( x = 1);
        printf(" x equals 1" );
    otherwise
        printf(" x does not equal 1");
    return 0;
}
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

TYPE & RUN 2　猜数游戏

这是第二个 Type & Run。Type & Run 旨在提供一些功能比各天课程中的程序清单更强大些的程序。该程序清单包含一些本书还未介绍过的内容，但您将发现它易于理解。输入并运行该程序后，花一些时间对其中的代码进行试验。对程序进行修改，然后重新编译和运行它们，看看会出现什么情况。如果出现错误，请确保您正确地输入了该程序清单。

程序清单 T&R 2　　find_nbr.c

```
1:  /* Name:     find_nbr.c
2:   * Purpose:  This program picks a random number and then
3:   *           lets the user try to guess it
4:   * Returns:  Nothing
5:   */
6:
7:  #include <stdio.h>
8:  #include <stdlib.h>
9:  #include <time.h>
10:
11: #define NO   0
12: #define YES  1
13:
14: int main( void )
15: {
15:     int guess_value = -1;
17:     int number;
18:     int nbr_of_guesses;
19:     int done = NO;
20:
21:     printf("\n\nGetting a Random number\n");
22:
23:     /* use the time to seed the random number generator */
24:     srand( (unsigned) time( NULL ) );
25:     number = rand();
26:
27:     nbr_of_guesses = 0;
28:     while ( done == NO )
29:     {
30:         printf("\nPick a number between 0 and %d> ", RAND_MAX);
31:         scanf( "%d", &guess_value );  /* Get a number */
32:
33:         nbr_of_guesses++;
34:
```

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
35:         if ( number == guess_value )
36:         {
37:             done = YES;
38:         }
39:         else
40:         if ( number < guess_value )
41:         {
42:             printf("\nYou guessed high!");
43:         }
44:         else
45:         {
46:             printf("\nYou guessed low!");
47:         }
48:     }
49:
50:     printf("\n\nCongratulations! You guessed right in %d Guesses!",
51:             nbr_of_guesses);
52:     printf("\n\nThe number was %d\n\n", number);
53:
54:     return 0;
55: }
```

该程序是一个简单的猜数游戏。您要找出计算机随机生成的数字。每当您做出猜测后，计算机都将指出是大了还是小了。当您猜对后，计算机将祝贺您，并告诉您一共猜了多少次。

如果您想作弊，可以在程序中添加一行，打印出计算机生成的随机数。您可以在第二次编译该程序之前，添加下列代码：

```
26: printf( "The random number (answer) is: %d", number ); /* cheat */
```

这将让您知道，程序在正确运行。如果您决定将该程序提供给朋友，请务必删除上述作弊的代码。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第 5 天课程　使用函数封装代码

函数是C语言编程和C程序设计理念的核心。前面介绍了一些库函数，它们是由编译器提供的；今天的课程将介绍用户定义的函数。顾名思义，用户定义的函数是由程序员定义并创建的。今天将介绍以下内容：

- 函数及其组成；
- 使用函数的结构化编程的优点；
- 如何创建函数？
- 如何在函数中声明局部变量？
- 如何从函数将值返回给程序？
- 如何给函数传递参数？

5.1　函数是什么

今天将以两种方式回答“函数是什么”这一问题。首先介绍函数是什么，然后介绍如何使用它们。

5.1.1　函数的定义

函数的定义为：函数是一个被命名的、独立的代码段，它执行特定的任务，并可能给调用它的程序返回一个值。该定义中的各部分如下：

- 函数是被命名的。每个函数都有唯一的名称，在程序的其他部分使用该名称，可以执行函数中的语句。这被称为调用函数，可以在一个函数中调用另一个函数。
- 函数是独立的、无需程序其他部分的干预，函数便能够执行其任务。
- 函数执行特定的任务。任务是程序运行时必须执行的具体工作，如将一行文本发送给打印机、对数组进行排序、计算立方根等。
- 函数可以将一个值返回给调用它的程序。程序调用函数时，将执行该函数中的语句，而这些语句可以将信息返回给调用它们的程序。

这便是有关函数定义的所有内容。阅读下一节时，请记住上述定义。

5.1.2　函数的用法

程序清单 5.1 包含一个用户定义的函数。

程序清单 5.1　　cube.c：使用函数来计算数的立方

```
1:  /* Demonstrates a simple function */
2:  #include <stdio.h>
3:
4:  long cube(long x);
5:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
6:  long input, answer;
7:
8:  int main( void )
9:  {
10:    printf("Enter an integer value: ");
11:    scanf("%d", &input);
12:    answer = cube(input);
13:    /* Note: %ld is the conversion specifier for */
14:    /* a long integer */
15:    printf("\nThe cube of %ld is %ld.\n", input, answer);
16:
17:    return 0;
18: }
19:
20: /* Function: cube() - Calculates the cubed value of a variable */
21: long cube(long x)
22: {
23:    long x_cubed;
24:
25:    x_cubed = x * x * x;
26:    return x_cubed;
27: }
```

该程序的运行情况如下：

```
Enter an integer value: 100

The cube of 100 is 1000000.
Enter an integer value: 9

The cube of 9 is 729.
Enter an integer value: 3

The cube of 3    is 27.
```



注意：下面的分析侧重于程序中直接与函数相关的部分，而不解释整个程序。

分析：第 4 行包含函数原型——程序后面将出现的函数的模型。函数原型包含函数名称、传递给函数的变量列表以及函数返回的变量的类型。从第 4 行可以知道，该函数名为 cube，它接受一个 long 变量，并返回一个类型为 long 的值。传递给函数的变量被称为参数，位于函数名后面，并用圆括号括起。这里，函数只有一个参数：long x。函数名前面的关键字指定了函数返回的变量的数据类型，这里为 long。

第 12 行调用 cube，并将变量 input 作为参数传递给它。函数的返回值被赋给变量 answer。在第 6 行，变量 input 和 answer 都被声明为 long 类型，这与第 4 行的函数原型使用的类型一致。

函数本身为函数定义。这里，函数名为 cube，位于第 21～27 行。和原型一样，函数定义也由几部分组成。函数定义以函数头（第 21 行）开始，它指定了函数的名称（cube）、返回类型和参数。函数头与函数原型相同，只是没有分号。

函数体（第 22～27 行）用花括号括起，其中包含函数被调用时将执行的语句（如第 25 行）。第 23 行是一个变量声明，它与以前介绍的变量声明类似，但有一点不同：该变量为局部变量。局部变量是在函数体中声明的，这将在第 12 天的课程中做进一步的讨论。最后，函数以 return 语句（第 26 行）结束。return 语句标

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

记函数结束，同时可以将一个值传递给调用函数的程序。这里返回的是变量 x_cubed 的值。

比较函数 cube()和 main()可以发现，它们的结构完全相同。读者前面使用了 printf()和 scanf()函数，虽然它们是库函数（而不是用户定义的函数），但和用户创建的函数一样，也可以接受参数并返回值。

5.2　函数的工作原理

仅当函数被程序的其他部分调用后，函数中的语句才会被执行。调用函数时，程序可以通过一个或多个参数给它传递信息。参数是程序传递给函数的数据，函数可以使用这些数据执行任务。然后执行函数中的语句，完成被设计的任务。函数中的语句执行完毕后，控制权将返回到调用函数的地方。函数能够以返回值的方式将信息返回给程序。

图 5.1 是一个包含三个函数的程序，其中每个函数都被调用一次。每当函数被调用时，控制权便被传递给函数。函数执行完毕后，控制权返回到调用该函数的位置。可以以任何顺序调用函数任意次。

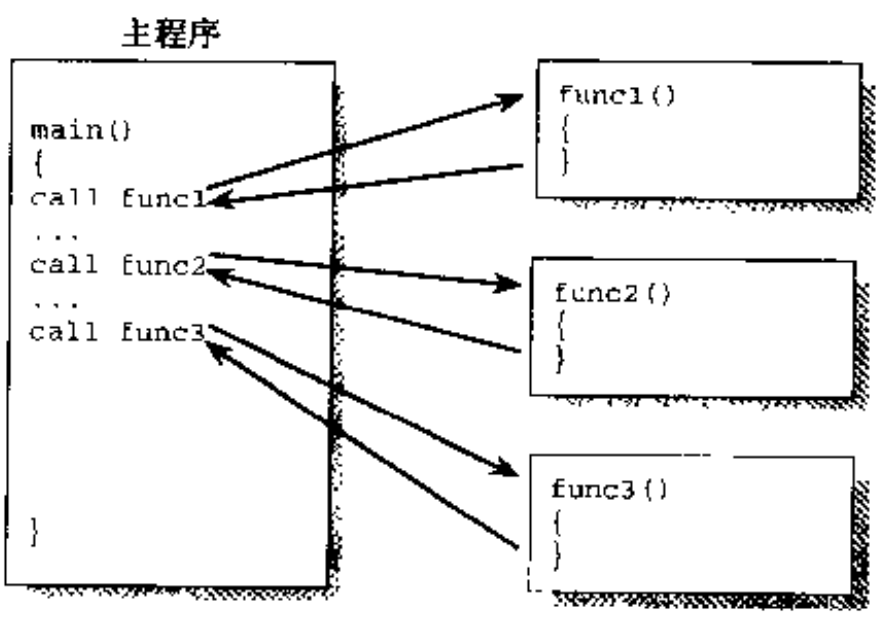


图 5.1　程序调用函数时，控制权将转到函数，然后再返回到程序

至此您已了解函数是什么以及函数的重要性，接下来将介绍如何编写和使用函数。

函数原型的格式如下：

```
return-type function_name( arg-type name-1,...,arg-type name-n);
```

函数定义的格式如下：

```
return-type function_name( arg-type name-1,...,arg-type name-n)
{
    /* statements; */
}
```

函数原型使编译器能够了解后面要定义的函数。原型中包含返回类型（指出函数将返回的变量的类型）、函数名称（描述函数的功能）以及传递给函数的参数的类型（arg-type），还可以包含传递给函数的变量的名称。函数原型总是以分号结尾。

函数定义是实际的函数，其中包含要执行的代码。如果函数原型中包含变量名，则函数定义的第一行（函数头）必须与函数原型相同，只是没有分号。函数头不应以分号结尾。另外，虽然在函数原型中，参数变量名是可选的，但函数头中必须包含。函数头的后面是函数体，其中包含函数将执行的语句。函数体以左花括号开始，以右花括号结束。如果函数的返回类型不是 void，则函数体必须包含一条 return 语句，它返回一个类型为返回类型的值。

下面是几个函数原型的例子：

```
double squared( double number );
void print_report( int report_number );
int get_menu_choice( void);
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

下面是几个函数定义的例子：

```
double squared( double number )          /* function header */
{                                        /* opening bracket */
        return( number * number );      /* function body   */
}                                       /* closing bracket */
void print_report( int report_number )
{
        if( report_number == 1 )
            puts( "Printing Report 1" );
        else
            puts( "Not printing Report 1" );
}
```

5.3 函数和结构化编程

通过在程序中使用函数，可以进行结构化编程。在结构化编程中，各个任务是由独立的程序代码段完成的。“独立的程序代码段”与函数概念中的内容类似，不是吗？函数和结构化编程关系紧密。

5.3.1 结构化编程的优点

结构化编程之所以卓越，有两个重要原因：

- 结构化程序更容易编写，因为复杂的编程问题被划分为多个更小、更简单的任务。每个任务由一个函数完成，而函数中的代码和变量独立于程序的其他部分。通过每次处理一个相对简单的任务，编程速度将更快。
- 结构化程序更容易调试。如果程序中有 bug（导致程序无法正确运行的东西），结构化设计则使得将问题缩小到特定的代码段（如特定的函数）更容易。

结构化编程的一个相关优点是可以节省时间。如果您在一个程序中编写了一个执行特定任务的函数，则可以在另一个需要执行相同任务的程序中使用它。即使新程序需要完成的任务稍微不同，但修改一个已有的函数比重新编写一个新函数更容易。想想看，您经常使用函数 printf()和 scanf()，虽然您可能还不知道它们的代码。如果您的函数用于执行单个任务，则在其他程序中使用它将容易得多。

5.3.2 规划结构化程序

编写结构化程序之前，必须做一些规划。规划必须在编写代码前完成，通常这只需使用笔和纸便可完成。规划中必须列出程序要执行的所有具体任务。首先应确定程序的功能。如果要编写的是管理联系地址的程序（姓名和地址列表），您希望该程序具备哪些功能。下面是一些显而易见的功能：

- 输入新的姓名和地址；
- 修改已有的条目；
- 按姓对条目进行排序；
- 打印邮寄地址标签。

根据上述列表，便可以将程序分为4个主要任务，其中每个任务用一个函数来完成。现在再进一步将这些任务分别划分为更小的子任务。例如，可以将任务“输入新的姓名和地址”划分为以下子任务：

- 从磁盘中读取已有的地址列表；
- 提示用户输入一个或多个条目；
- 将新数据添加到列表中；
- 将更新后的列表存盘。

同样，任务“修改已有的条目”也可以划分为以下的子任务：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

- 从磁盘中读取已有的地址列表；
- 修改一个或多个条目；
- 将更新后的列表存盘。

您可能注意到了，上述两个列表有两个相同的子任务：读取磁盘和存盘。可以为“从磁盘读取已有的地址列表”编写一个函数，然后在函数“输入新的姓名和地址”和“修改已有的条目”中调用这个函数。对于“将更新后的列表存盘”，也可以做相同的处理。

现在，您至少明白了结构化编程的一种优点。通过将程序划分为不同的任务，可以发现程序中需要完成相同任务的部分。您可以编写存储磁盘的函数，然后多次调用它们，这样可以节省时间，并使程序更小、效率更高。

这种编程方法编写出的程序为层次化结构的，图 5.2 说明了地址列表程序的层次化结构。

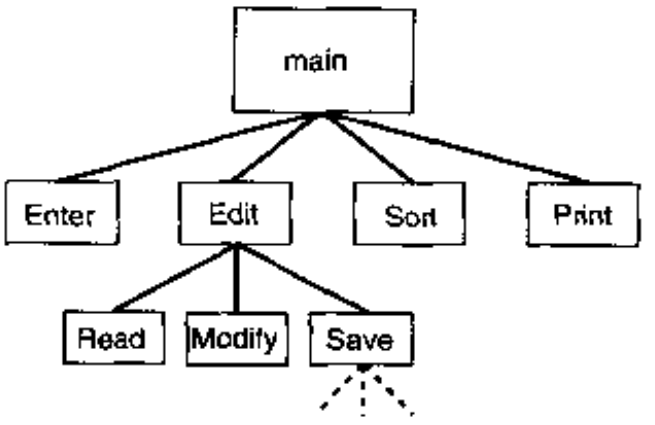


图 5.2　以层次方式组织的结构化程序

按照这种规划方式，很快便能列出程序需要执行的具体任务。然后，每次处理一个任务，将全部精力放在一个相对简单的任务上。函数编写好并能正确工作后，便可以进入到下一个任务。不知不觉中，程序便成型了。

5.3.3　从顶向下的方法

采用结构化编程时，C 程序员可以选用从顶向下的方法。图 5.2 说明了这一点，在该图中程序的结构就像一个倒立的树。很多情况下，程序的大多数实际工作是由位于树枝末梢的函数完成的，位于“主干”附近的函数主要用于引导程序执行这些函数。

因此，很多 C 程序的主体——main()包含的代码很少，程序的大部分代码位于函数中，main()中只有几十行代码，用于引导程序执行函数。通常，给程序用户提供了一个菜单，根据用户的选择执行程序的分支，菜单的每个分支使用不同的函数。



注意：使用菜单是一种不错的程序设计方式。第 13 天的课程将介绍如何使用 switch 语句创建一个菜单驱动系统。

至此，您已了解什么是函数以及它为什么如此重要，接下来介绍如何编写自己的函数。

应　读	不 应 读
编写代码之前一定要进行规划，预先确定程序的结构可以节省编写和调试代码的时间。	不要将所有的功能放在一个函数中。一个函数只应完成一项任务，如读取文件中的信息。

5.4　编写函数

编写函数时首先要明白您希望函数做什么。知道这一点后，编写起来便不会太困难。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

5.4.1 函数头

每个函数的第一行都是函数头，函数头由三部分组成，其中每一部分完成特定的功能。图 5.3 说明了这三部分，接下来的几节将分别介绍它们。

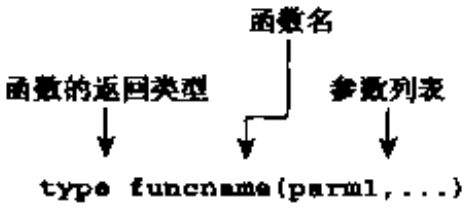


图 5.3 函数头的三个组成部分

5.4.2 函数的返回类型

函数的返回类型指定了函数返回给调用程序的数据类型。函数的返回类型可以是任何数据类型，包括 char、int、long、float 或 double。可以使用返回类型 void 来指定函数不返回任何值。下面是一些范例：

```
int func1(...)          /* Returns a type int.    */
float func2(...)        /* Returns a type float.  */
void func3(...)         /* Returns nothing.       */
```

其中 func1 返回一个整数，func2 返回一个浮点数，而 func3 不返回任何值。

5.4.3 函数名

您可以将函数命名为任何名称，只要遵循变量名规则（参见第 3 天的课程）即可。在 C 程序中，函数名必须是唯一的（与其他函数或变量的名称不同）。给函数指定一个描述其功能的名称是个不错的主意。

5.4.4 参数列表

很多函数使用参数——调用函数时传递给它的值。函数需要知道它期望什么样的参数——每个参数的数据类型。可以给函数传递任何数据类型，参数类型信息是由函数头中的参数列表提供的。

对于要传递给函数的每个参数，参数列表中必须包含一个相应的条目。该条目指定参数的数据类型和名称。例如，下面是程序清单 5.1 中的一个函数的函数头：

```
long cube(long x)
```

其中参数列表为 long x，它指定该函数接受一个类型为 long 的参数，该参数用 x 表示。如果有多个参数，必须使用逗号将各个参数分开。下面的函数头：

```
void func1(int x, float y, char z)
```

定义了一个接受三个参数的函数：一个名为 x，类型为 int；一个名为 y，类型为 float；一个名为 z，类型为 char。有些函数不接受任何参数，在这种情况下，参数列表为 void，如下所示：

```
int func2(void)
```



注意：不能在函数头后面加上分号，如果这样做，编译器将产生错误消息。

有时候人们会混淆形参（parameter）和实参（argument）。形参位于函数头中，是实参的一个占位符。函数的形参是固定的，在程序执行期间不会变化。

实参是调用程序传递给函数的实际值。每次调用函数时，可以传递不同的实参。在 C 语言中，每次调用函数时，传递的实参的个数和类型必须相同，但实参的值可以不同。在函数中，通过相应的形参名来访问实参。

下面的范例更清楚地说明了这一点。程序清单 5.2 是一个非常简单的程序，它包含一个函数，该函数被调用两次。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 5.2　　list0502.c：形参和实参之间的区别

```
1:  /* Illustrates the difference between arguments and parameters. */
2:
3:  #include <stdio.h>
4:
5:  float x = 3.5, y = 65.11, z;
6:
7:  float half_of(float k);
8:
9:  int main( void )
10: {
11:     /* In this call, x is the argument to half_of(). */
12:     z = half_of(x);
13:     printf("The value of z = %f\n", z);
14:
15:     /* In this call, y is the argument to half_of(). */
16:     z = half_of(y);
17:     printf("The value of z = %f\n", z);
18:
19:     return 0;
20: }
21:
22: float half_of(float k)
23: {
24:     /* k is the parameter. Each time half_of() is called, */
25:     /* k has the value that was passed as an argument. */
26:
27:     return (k/2);
28: }
```

该程序的输出如下：

```
The value of z = 1.750000
The value of z = 32.555000
```

图 5.4 说明了形参和实参之间的关系。

分析：从程序清单 5.2 可知，第 7 行声明了函数 half_of()的原型。第 12 和 16 行调用了 half_of()函数，而该函数的定义位于第 22～28 行。第 12 和 16 行分别将不同的实参传递给函数 half_of()，前者传递的是变量 x（值为 3.5），后者传递的是变量 y（值为 65.11）。程序运行时，分别打印了正确的值。变量 x 和 y 的值分别被传递给函数 half_of()的形参 k，这相当于将 x 和 y 的值复制给 k。然后，函数 half_of()分别将这些值除以 2，并返回得到的结果（第 27 行）。

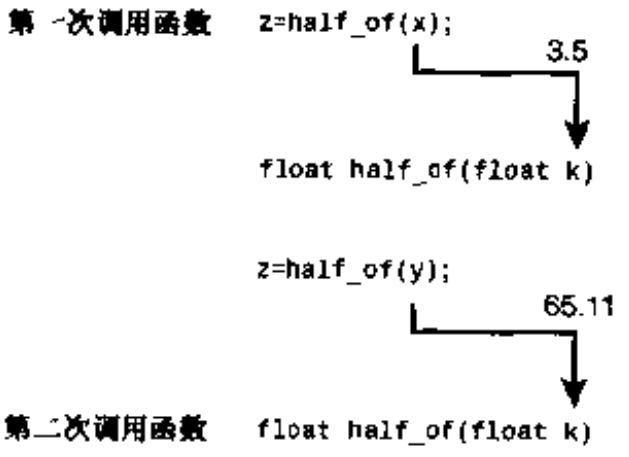


图 5.4　调用函数时，实参被传递给形参

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

应　该	不应该
应给函数指定一个描述其用途的名称。 应确保传递给函数的实参的数据类型与函数的形参匹配。	不要将不必要的值传递给函数。 传递给函数的实参数目不应少于（或多于）形参数目。在 C 程序中，传递给函数的实参数必须和形参数相等。

5.4.5　函数体

函数体位于函数头后面，用花括号括起。实际的工作是在函数体中完成的。函数被调用后，首先执行函数体中的第一条语句，执行到 return 语句或最外面的花括号后结束（返回到调用程序）。

1．局部变量

可以在函数体中声明变量，这种变量被称为局部变量。“局部”意味着变量是特定函数私有的，不同于程序其他地方声明的同名变量。稍后将对此进行解释，现在介绍如何声明局部变量。

声明局部变量的方式和其他变量相同，可以使用第 3 天课程介绍的变量类型和命名规则。也可以在声明时初始化局部变量。在函数中，可以声明任何类型的变量。下面的范例在一个函数中声明了 4 个局部变量：

```
int func1(int y)
{
    int a, b = 10;
    float rate;
    double cost = 12.55;
    /* function code goes here... */
}
```

上述声明创建了 4 个局部变量：a、b、rate 和 cost，函数中的代码可以使用这些变量。注意，函数形参被视为变量声明，因此函数的参数列表中的变量也是可用的。

在函数中声明和使用变量时，该变量与程序的其他地方声明的变量是完全不同的，即使它们的名称相同。程序清单 5.3 说明了这一点。

程序清单 5.3　　var.c：演示局部变量

```
1:  /* Demonstrates local variables. */
2:
3:  #include <stdio.h>
4:
5:  int x = 1, y = 2;
6:
7:  void demo(void);
8:
9:  int main( void )
10: {
11:   printf("\nBefore calling demo(), x = %d and y = %d.", x, y);
12:   demo();
13:   printf("\nAfter calling demo(), x = %d and y = %d\n.", x, y);
14:
15:   return 0;
16: }
17:
18: void demo(void)
19: {
20:     /* Declare and initialize two local variables. */
21:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
22:     int x = 88, y = 99;
23:
24:     /* Display their values. */
25:
26:     printf("\nWithin demo(), x = %d and y   %d.", x, y);
27: }
```

该程序的输出如下：

```
Before calling demo(), x = 1 and y = 2.
Within demo(), x = 88 and y = 99.
After calling demo(), x = 1 and y = 2.
```

分析：程序清单 5.3 同今天课程中的前几个程序有些类似。第 5 行声明了变量 x 和 y，它们是在函数的外面被声明的，因此为全局变量。第 7 行是演示函数 demo()的原型，它不接受任何参数，因此原型中包含 void；同时也不返回任何值，因此返回类型为 void。从第 9 行开始是 main()函数，它非常简单。首先，第 11 行调用函数 printf()以显示 x 和 y 的值，然后调用 demo()函数。在第 22 行，demo()声明了自己的局部变量 x 和 y。第 26 行表明，局部变量优先于其他变量。调用 demo()函数后，第 13 行再次打印了 x 和 y 的值。由于此时不在 demo()函数中，因此打印的是全局变量的值。

正如您看到的，函数中的局部变量 x 和 y 完全独立于函数外面声明的全局变量 x 和 y。在函数中使用变量的规则如下：

- 要在函数中使用变量，必须在函数头或函数体声明它（全局变量除外，这将在第 12 天的课程中介绍）；
- 函数要从调用程序那里获得值，必须将这个值以实参的形式传递给函数；
- 调用程序要从函数那里获得值，函数必须显式地返回它。

坦率地说，这些规则并没有严格地应用，因为本书的后面将介绍如何避开它们。然而，现在只需遵循这些规则即可，这样可以避免麻烦。

让函数的变量独立于程序的变量是函数独立的途径之一。使用自己的一组变量，函数可以对数据执行任何操作，而不会无意间影响到程序的其他部分。

2. 函数的语句

函数几乎可以包含任何语句，在函数中唯一不能做的是定义另一个函数，但可以使用其他任何语句，包括循环（将在第 6 天的课程中介绍）、if 语句和赋值语句，还可以调用库函数和其他用户定义的函数。

函数的长度方面呢？C 语言对函数的长度没有任何限制，但出于实用的考虑，您应使函数比较简短。在结构化编程中，每个函数都只应完成一个相对简单的任务。如果函数很长，则很可能是由于您在该函数中完成的任务过于复杂，也许可以将其划分为两个或更多的函数。

多长是太长了呢？这个问题没有明确的答案，但实践经验表明，函数的实际代码很少有超过 25～30 行的。如果超过这样的长度，则说明函数很可能执行了多项任务。您必须自己做出判断。有些编程任务需要使用较长的函数，而很多函数则只有几行代码。随着编程经验的日渐丰富，对于是否需要将函数划分为更小的函数，您将会越来越得心应手。

3. 返回一个值

要从函数返回一个值，可以使用关键字 return，并在后面加上一个表达式。程序执行到 return 语句时，该表达式将被计算，然后返回到调用程序处继续执行。函数的返回值为该表达式的值。请看下面的函数：

```
int func1(int var)
{
    int x;
    /* Function code goes here... */
    return x;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
}
```

当上述函数被调用时，该函数体中的语句将被执行，直到 return 语句。return 语句结束函数，并将 x 的值返回给调用程序。关键字 return 后面的表达式可以是任何合法的表达式。

函数可以包含多条 return 语句，但只有第一条被执行的 return 语句对程序有影响。使用多条 return 语句是一种高效的、从函数返回不同值的方式，程序清单 5.4 演示了这一点。

程序清单 5.4　　return.c：在函数中使用多条 return 语句

```
1:  /* Demonstrates using multiple return statements in a function. */
2:
3:  #include <stdic.h>
4:
5:  int x, y, z;
6:
7:  int larger_of( int a, int b);
8:
9:  int main( void )
10: {
11:     puts("Enter two different integer values: ");
12:     scanf("%d%d", &x, &y);
13:
14:     z = larger_of(x,y);
15:
16:     printf("\nThe larger value is %d.", z);
17:
18:     return 0;
19: }
20:
21: int larger_of( int a, int b)
22: {
23:     if (a > b)
24:        return a;
25:     else
26:        return b;
27: }
```

该程序的运行情况如下：

```
Enter two different integer values:
200 300

The larger value is 300.
```

分析：和其他程序清单一样，程序清单 5.4 的开头也是一条描述程序功能的注释。为了让程序能够使用标准输入/输出函数将信息显示到屏幕上并读取用户的输入，包含了头文件 stdio.h。第 7 行是函数 larger_of() 的原型，该函数接受两个 int 参数，并返回一个 int 值。第 14 行调用 larger_of()，并将 x 和 y 传递给它。这个函数包含多条 return 语句。在第 23 行，函数使用 if 语句判断 a 是否大于 b。如果 a 大于 b，则执行第 24 行的 return 语句，然后函数立刻结束。在这种情况下，第 25 和 26 行将不会执行。如果 a 不大于 b，则跳过第 24 行，执行第 26 行的 return 语句。从中可以知道，两条 return 中只有一条被执行，并将合适的值返回给调用函数，到底哪条 return 语句被执行取决于传递给函数 larger_of()的实参。

最后需要指出的一点是，第 11 行是一个以前没有介绍过的新函数——puts()，它将一个字符串显示到标准输出——通常是计算机屏幕（字符串将在第 10 天的课程中介绍，就现在而言，您只需知道它们是用引号括

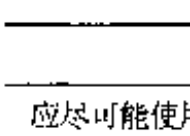
鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

起的文本即可）。

函数的返回值的类型是在函数头和函数原型中指定的。函数返回的值的类型必须和指定的相同，否则编译器将生成错误消息。



注意：结构化编程建议函数只有一个入口和一个出口。这意味着函数应尽量只包含一条 return 语句，然而有时候，使用多条 return 语句时，程序将更容易理解和维护。在这种情况下，应优先考虑可维护性。

5.4.6　函数原型

对于其使用的每个函数，程序都应包含一个原型。程序清单 5.1 中的第 4 行便是一个函数原型，其他程序清单中也有函数原型。函数原型是什么？为何它是必不可少的？

从前面的范例可知，函数原型和函数头相同，只是后面多了一个分号。和函数头一样，函数原型也包含了有关函数的返回类型、名称和参数的信息。函数原型的功能是将有关函数的信息告知编译器。知道有关函数的返回类型、名称和参数后，编译器便可以检查源代码中每次对函数的调用，确保您传递的参数数目和类型是正确的，并确保你正确地使用了返回值。如果存在不匹配的地方，编译器将生成错误消息。

严格地讲，函数原型并不必与函数头完全相同。参数的名称可以不同，只要参数的类型、数目和顺序相同即可。没有理由让函数头和函数原型不同，它们相同可使源代码更易于理解，也使编写程序时更容易。编写好函数定义后，可以使用编辑器的剪切和粘贴特性来复制函数头并创建函数原型，但请务必在函数原型的后面加上分号。

应将函数原型放在源代码的什么位置呢？应放在第一个函数之前。为提高可读性，最好将所有的函数原型放在一起。

应　该	不 应 该
应尽可能使用局部变量； 每个函数只应完成一项任务。	不要返回类型不同于函数类型的值； 函数不应过长。当函数过长时，应将其划分为几个小任务。 应尽可能不使用多条 return 语句；但有时候使用多条 return 语句时，程序将更简单、更清晰。

5.5　将参数传递给函数

要将参数传递给函数，可将它们放在函数名的后面，并用圆括号括起。参数的数目与类型必须同函数头和函数原型中的形参匹配。例如，如果函数被定义为接受两个 int 参数，则必须给它传递两个 int 参数——不能多，不能少，也不能是其他类型。如果传递的参数数据和/或类型不正确，编译器将根据函数原型中的信息发现这一点。

如果函数接受多个参数，则函数调用中的参数将被依次赋给函数的形参：第一个参数被赋给第一个形参，第二个参数被赋给第二个形参，依次类推，如图 5.5 所示。

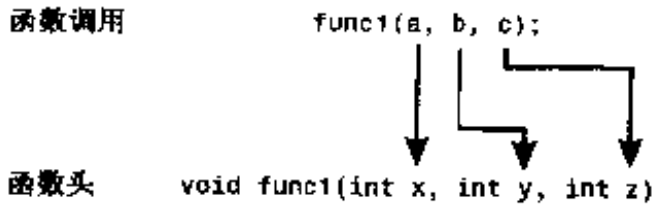


图 5.5　多个参数被依次赋给函数的形参

每个参数都可以是任何合法的表达式：常量、变量、数学或逻辑表达式，甚至可以是另一个函数（返回一个值的函数）。例如，如果函数 half()、square()和 third()都有返回值，则可以这样编写代码：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
x = half(third(square(half(y))));
```

程序首先调用 half()，并将 y 作为参数传递给它。从 half()返回后，程序调用 square()，并将 half()的返回值作为参数传递给它。接下来，third()被调用，并将 square()的返回值作为参数传递给它。然后，half()函数被再次调用，并将 third()的返回值作为参数传递给它。最后，half()的返回值被赋给变量 x。下述代码与此等价：

```
a = half(y);
b = square(a);
c = third(b);
x = half(c);
```

5.6 调用函数

调用函数的方式有两种。对于任何函数，都可以使用其名称和参数列表进行调用，如下面的范例所示。如果函数有返回值，则被丢弃。

```
wait(12);
```

第二种方法只能用于有返回值的函数。由于这些函数的结果为一个值（即返回值），因此是合法的表达式，可用于任何能使用表达式的地方。您已经见过将有返回值的函数放在赋值语句右边的情况。下面是其他一些例子。

在下面的范例中，half_of()被用作函数的参数：

```
printf("Half of %d is %d.", x, half_of(x));
```

首先将 x 的值作为参数调用 half_of()，然后将“Half of %d is %d.”、x 和 half_of(x)的返回值作为参数，调用 printf()。

在下面的范例中，在一个表达式中使用了多个函数：

```
y = half_of(x) + half_of(z);
```

虽然这里两次使用的都是 half_of()，但第二次也可以使用其他函数。下述代码的功能与此等效，但占多行：

```
a = half_of(x);
b = half_of(z);
y = a + b;
```

接下来的两个例子演示了如何高效地使用函数的返回值。下面的这个例子将函数用于 if 语句中：

```
if ( half_of(x) > 10 )
{
        /* statements; */          /* these could be any statements! */
}
```

如果函数的返回值符合条件（即如果 half_of()返回的值大于 10），则 if 语句为真，其中的语句将被执行；否则不执行。

下面的范例更好：

```
if ( do_a_process() != OKAY )
{
        /* statements; */          /* do error routine */
}
```

同样，这里没有提供实际的语句，do_a_process 也不是一个真正的函数，但这是一个非常重要的范例。它检查处理过程的返回值，以判断处理是否成功地完成。如果没有，则由后面的语句进行错误处理或清理工作。在存取文件中的信息、对值进行比较以及分配内存时，常常这样做。



警告：如果您把返回类型为 void 的函数用作表达式，编译器将生成错误消息。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

应　该	不应该
应给函数传递参数，以提高其通用性和可重用性。 应充分利用可将函数作为表达式的功能。	不要在一条语句中包含过多的函数，以免引起混淆；仅当不会引起混淆时，才应将函数放到语句中。

5.6.1　递归

递归指的是函数直接或间接地调用自己。间接递归指的是一个函数调用另一个函数，而后者又调用前者。C 语言允许递归，在有些情况下，递归很有用。

例如，递归可用于计算阶乘。x 的阶乘表示为 x!，计算方法如下：

```
x! = x * (x-1) * (x-2) * (x-3) * ... * (2) * 1
```

然而，也可以这样计算 x!：

```
x! = x * (x-1)!
```

然后，可以使用同样的方式计算 x-1 的阶乘：

```
(x-1)! = (x-1) * (x-2)!
```

您可以不断地以递归的方式计算下去，直到 1。程序清单 5.5 中的程序使用了一个递归函数来计算阶乘。由于该程序使用的是无符号整型变量，因此最大输入值为 8。当输入值为 9 时，阶乘将超出这种变量的取值范围。

程序清单 5.5　recurse.c：使用递归函数计算阶乘

```
1:  /* Demonstrates function recursion. Calculates the */
2:  /* factorial of a number. */
3:
4:  #include <stdio.h>
5:
6:  unsigned int f, x;
7:  unsigned int factorial(unsigned int a);
8:
9:  int main( void )
10: {
11:     puts("Enter an integer value between 1 and 8: ");
12:     scanf("%d", &x);
13:
14:     if( x > 8 || x < 1)
15:     {
16:        printf("Only values from 1 to 8 are acceptable!");
17:     }
18:     else
19:     {
20:        f = factorial(x);
21:        printf("%u factorial equals %u\n", x, f);
22:     }
23:
24:     return 0;
25: }
26:
27: unsigned int  factorial(unsigned int a)
28: {
29:     if (a == 1)
30:        return 1;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
31:     else
32:     {
33:        a *= factorial(a-1);
34:        return a;
35:     }
36: }
```

该程序的运行情况如下：

```
Enter an integer value between 1 and 8:
6
6 factorial equals 720
```

分析：该程序的前半部分与前面介绍的很多程序类似。第 1 和 2 行为注释，第 4 行包含了输入/输入函数所在的头文件。第 6 行声明了两个无符号整型变量，而第 7 行是计算阶乘的函数的原型，该函数接受一个 unsigned int 参数，并返回一个 unsigned int 值。第 9～25 行为 main()函数。第 11 和 12 行打印一条消息，提示用户输入一个 1～8 的值，然后读取用户输入的值。

第 14～22 行是一条有趣的 if 语句。由于用户输入的值大于 8 时，将导致问题，因此该 if 语句检查用户输入的值。如果它大于 8，则打印一条错误消息；否则计算其阶乘（第 20 行）并打印结果（第 21 行）。当您知道可能出现问题（如某个数字过大）时，请添加检测并防范这种问题的代码。

递归函数 factorial()位于第 27～36 行。传递的值被赋给变量 a。第 29 行检查 a 的值，如果为 1，则返回 1；否则将 a 与 factorial(a-1)的乘积赋给 a。程序将再次调用 factorial 函数，但这次 a 的值为 a-1。如果 a-1 不等于 1，则以（(a-1）-1）（即 a-2）为参数，再次调用 factorial。这一过程将一直重复下去，直到第 29 行的 if 语句为真。如果输入的值为 3，则阶乘为：

```
3 * (3-1) * ((3-1)-1)
```

应　该	不 应 该
在要分发的程序中使用递归之前，请务必理解递归。	如果将发生多次迭代（一次迭代相当于程序重复执行一次），请不要使用递归。递归将使用大量的资源，因为函数必须记住它所在的位置。

5.7　函数的位置

您可能会问，应将函数定义放置在源代码的什么位置。就现在而言，您应将其放在 main()函数所在的源代码文件中，并位于 main()函数的后面。图 5.6 说明了使用函数的程序的基本结构。

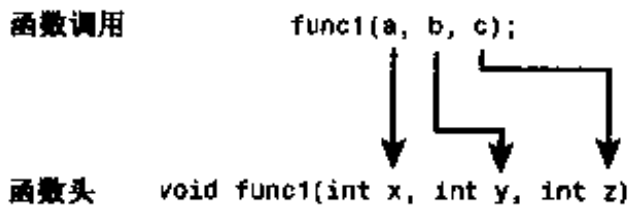


图 5.6　将函数原型放在函数定义之前。通常函数原型位于程序清单的开头

也可以将用户定义的函数和 main()函数放在不同的源代码文件中。当程序很大或要在多个程序中使用同一组函数时，这种技术很有用。有关这种技术的知识将在第 21 天的课程中介绍。

5.8　内联函数

在 C 语言中有一种特殊的函数——内联函数。旧版本的 C 语言大多数不支持内联函数，但遵循 C-99 标

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

准的编译器支持。

内联函数通常很短。编译器将尽可能使内联函数的执行速度最快，这可能通过将函数的代码复制到调用函数中来实现。由于将在调用函数中执行这种函数的代码，因此称为“内联”。

可以使用关键字 inline 来声明内联函数，下面的代码声明了一个名为 toInches 的内联函数：

```
inline int toInches( int Feet )
{
      return (Feet/12);
}
```

当 toInches()被使用时，编译器将尽可能对其进行优化，以提高其运行速度。虽然编译器通常会将代码复制到调用函数中，但并不一定总是这样。唯一可以肯定的是，编译器将尽可能对这种函数的代码进行优化。内联函数的用法与其他函数相同。

5.9　总　结

今天的课程介绍了函数——C 语言编程的一个重要组成部分。函数是执行特定任务的独立代码块。需要执行任务时，程序调用执行这种任务的函数。结构化编程（一种强调模块化的、自顶向下的程序设计方法）离不开函数。结构化编程能够创建高效的程序，程序员使用起来也很容易。

您还了解到，函数是由函数头和函数体组成的。函数头中包含有关函数的返回类型、名称和参数等信息；函数体中包含局部变量声明以及函数被调用时将执行的语句。最后，您知道局部变量（在函数内部声明的变量）完全独立于程序的其他地方所声明的变量。

5.10　问与答

问：如果要从函数返回多个值，该如何办？

答：很多时候，您需要从函数返回多个值，或者说您想修改传递给函数的值，并使函数结束后修改仍然有效。有关这一个主题将在第 18 天的课程中介绍。

问：什么是好的函数名？

答：好的函数名描述了函数的功能。

问：在 main()函数之前声明的变量是全局的，可以在程序的任何地方使用；而局部变量只能在特定的函数中使用。为何不在 main()函数之前声明所有的变量，使之为全局的？

答：有关变量的作用域将在第 12 天的课程中讨论，那时您将知道为何在函数中声明局部变量优于在 main()函数之前声明全局变量。

问：还有其他使用递归的方式吗？

答：阶乘函数是一个使用递归的绝佳范例。在很多统计计算中，需要使用阶乘。递归只不过是一种循环而已，但它有一个不同于循环的地方。使用递归时，每调用一次递归函数，都将创建一组新的变量；而对于下一章将介绍的其他循环来说，情况并非如此。

问：程序中的第一个函数必须是 main()吗？

答：不一定。C 语言规定，首先执行 main()函数，但它可以放在源代码文件的任何位置。大多数人将它放在最前面或最后面，以便容易找到。

问：什么是成员函数？

答：成员函数是诸如 C#、C++和 Java 等面向对象语言使用的一种特殊函数，它们位于类中。类是面向对象语言使用的一种特殊结构。有关成员函数的更详细的信息，请参阅附加课程。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

9. 编写一个函数，它调用练习 7 和练习 8 中的函数。

10. 编写一个程序，该程序使用一个函数来计算用户输入的 5 个 float 值的平均值。

11. 编写一个递归函数，计算 3 的 n 次幂，其中 n 为传递给该函数的整型参数。例如，如果传递的参数为 4，则该函数返回 81。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 6 天课程　基本的程序流程控制

第 4 天的课程介绍了让您能够控制程序流程的 if 语句。但很多时候，您需要的不仅仅是真或假的判断。今天的课程将介绍三种新的控制程序流程的方式。今天介绍以下内容：

- 如何使用简单数组？
- 如何使用 for、while 和 do...while 循环来多次执行语句？
- 如何嵌套程序控制语句？

今天的课程并不会介绍有关这些主题的所有内容，但将提供给您开始编写真正的程序所需的信息。第 13 天的课程将更详细地介绍这些主题。

6.1　数组的基本知识

介绍 for 语句之前，先介绍一些有关数组的基本知识（有关数组的完整介绍，见第 8 天的课程）。在 C 语言中，for 语句与数组密切相关，因此要解释其中的一个概念，如果不涉及另一个概念将很困难。为帮助您理解后面的 for 语句中使用的数组，这里对数组做一简要的介绍。

数组是一个包含索引的数据存储位置，它们的名称相同，通过下标或索引彼此区分开来。下标位于变量名的后面，用方括号括起。和其他变量一样，数组也必须声明。数组声明中包含数据类型和数组的长度（数组中的元素数）。例如，下面的语句声明了一个名为 data 的数组，该数组的类型为 int，包含 1000 个元素：

```
int data[1000];
```

通过下标来引用该数组中的元素，它们为 data[0]到 data[999]。第一个元素为 data[0]，而不是 data[1]。在其他语言（如 BASIC）中，数组的第一个元素的下标可能为 1，但在 C 语言中，情况并非如此。在 C 语言中，第一个元素的索引为 0。



> 注意：看待索引值的方式之一是将其视为偏移量。对于数组的第一个元素，偏移量为 0；对于第二个元素，其偏移量为 1，因此索引值为 1。

上述数组中的每个元素都相当于一个 int 变量，可以像使用 int 变量那样使用它们。数组的下标可以是另一个变量，如下面的范例所示：

```
int data[1000];
int index;
index = 100;
data[index] = 12;     /* The same as data[100] = 12 */
```

至此，对数组做了简要的介绍。现在，您将能够理解本章后面的程序范例是如何使用数组的。如果您还不清楚有关数组的所有细节，请不用担心，第 8 天的课程将更详细地介绍数组。

应　该	不 应 该
	声明数组时，下标不应超出您的需求，否则会浪费内存。 请别忘了，在 C 语言中，数组第一个元素的下标为 0，而不是 1。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

6.2　控制程序的执行

默认情况下，C 程序是自顶向下依次执行的。从 main()的起始位置开始，逐条地执行语句，直到 main()函数的最后。然而，在实际的 C 程序中，情况很少如此。C 语言提供了各种程序控制语句，使您能够控制程序的执行次序。前面介绍了如何使用判断运算符——if 语句，下面介绍其他三种很有用的控制语句：

- for 语句；
- while 语句；
- do...while 语句。

6.2.1　for 语句

for 语句是一种 C 编程结构，它将一个由一条或多条语句组成的代码块执行特定的次数。它有时候也被称为 for 循环，因为程序通常循环执行这种语句多次。本书前面的编程范例中使用过一些 for 语句。下面介绍 for 语句的工作原理：

for 语句的结构如下：

```
for ( initial; condition; increment )
    statement;
```

其中 initial、condition 和 increment 都是表达式，而 statement 为单条语句或复合语句。程序执行到 for 语句时，将发生以下事件：

1. 执行表达式 initial。initial 通常是一条赋值语句，将 一个变量设置为特定的值。
2. 判断 condition。condition 通常是一个关系表达式。
3. 如果 condition 为假（即等于 0），for 语句结束，并接着执行 statement 语句后面的第一条语句。
4. 如果 condition 为真（即不等于 0），则执行 statement 语句。
5. 执行表达式 increment，然后返回到第 2 步。

图 6.1 说明了 for 语句的工作原理。注意，如果首次计算时，condition 为 false，则 statement 一次也不会执行。

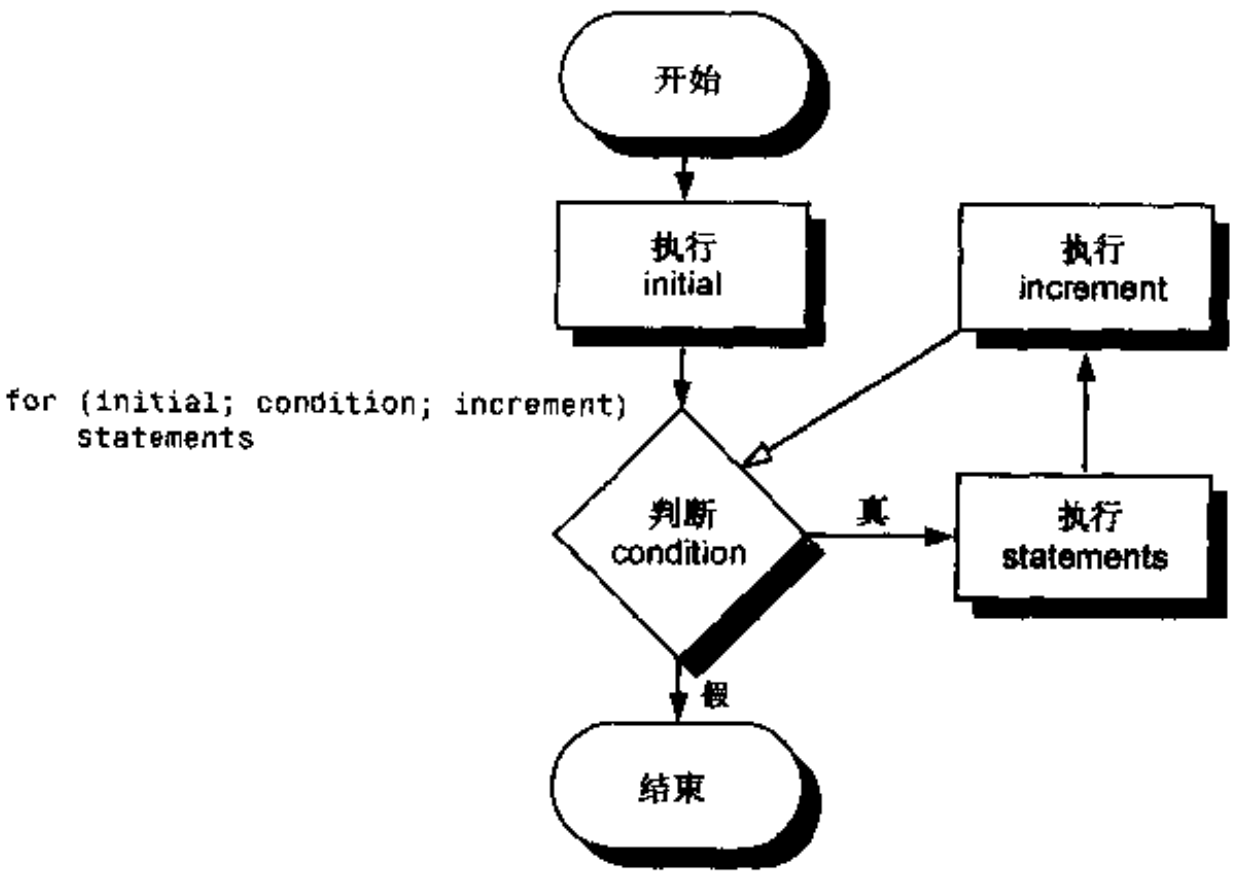


图 6.1　for 语句工作原理示意图

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 6.1 是一个简单的例子，它使用 for 语句来打印数字 1～20。从中可以知道，与对于 20 个值中的每个值，都分别使用一条 printf()语句相比，代码简短的多。

程序清单 6.1　　forstate.c：一条简单的 for 语句

```
1:  /* Demonstrates a simple for statement */
2:
3:  #include <stdio.h>
4:
5:  int count;
6:
7:  int main( void )
8:  {
9:      /* Print the numbers 1 through 20 */
10:
11:     for (count = 1; count <= 20; count++)
12:         printf("%d\n", count);
13:
14:     return 0;
15: }
```

该程序的输出如下：

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

图 6.2 说明了程序清单 6.1 中的 for 循环的运行过程。

分析：第 3 行包含了标准输入/输出头文件。第 5 行声明了一个名为 count 的 int 变量，供 for 循环使用。第 11 和 12 行是 for 循环。执行到 for 语句后，首先执行初始化语句。在该程序清单中，初始化语句为 count = 1，它初始化 count，以便 for 循环的其他部分能够使用该变量。接下来，对条件 count <= 20 进行判断。由于刚才已经将 count 初始化为 1，因此它小于 20，所以 for 命令中的语句 printf()将被执行。执行该打印函数后，将执行递增表达式 count++。这将 count 加 1，使其等于 2。接下来，程序回到 for 语句的开头，再次检查条件。如果为真，则再次执行 printf()，将 count 加 1（使之为 3），并再次检查条件。这一过程将不断循环下去，直到条件为假。然后程序将退出循环，并继续执行接下来的代码行（第 14 行）——在程序结束前将 0 返回。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

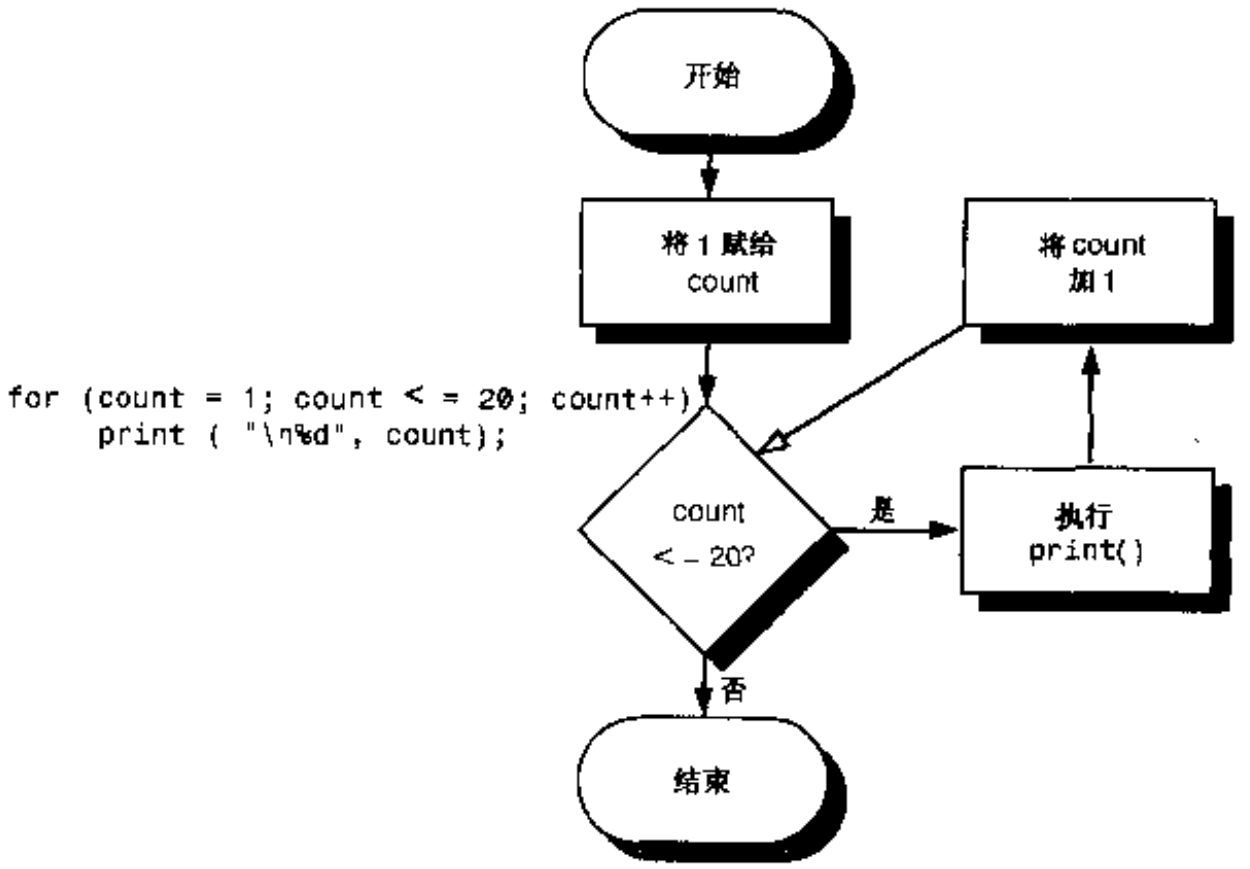


图 6.2　程序清单 6.1 中的 for 循环的运行过程

for 语句常被用来“向上计数”——将计数器从一个值增加到另一个值，如前面的范例所示。也可以使用它来“倒计数”——将计数器变量递减（而不是递增）。

```
for (count = 100; count > 0; count--)
```

也可以向下面的例子那样，不将递增量设置为 1（而是 5）：

```
for (count = 0; count < 1000; count += 5)
```

for 语句非常灵活。例如，如果已经在程序前面初始化了计数器变量，则可以省略初始化表达式，但不能省略分隔符分号：

```
count = 1;
for ( ; count < 1000; count++)
```

初始化表达式不必进行真正的初始化，而可以是任何表达式。但不管它是什么，都只会在首次到达 for 语句时被执行一次。例如，下面的初始化表达式代码打印“Now sorting the array...”：

```
count = 1;
for (printf("Now sorting the array...") ; count < 1000; count++)
    /* Sorting statements here */
```

也可以省略递增表达式，而在 for 语句体中执行递增操作。同样，分号也不能省略。例如，要打印 0～99 的数字，可以这样编写代码：

```
for (count = 0; count < 100; )
    printf("%d", count++);
```

用于终止 for 循环的测试表达式也可以是任何表达式。只要该表达式为真（非零），for 语句便会继续执行。您可以使用逻辑运算符来构建复杂的测试表达式。例如，下面的 for 语句打印数组 array[]的元素，当打印完所有的元素或者遇到值为 0 的元素后，将结束打印：

```
for (count = 0; count < 1000 && array[count] != 0; count++)
    printf("%d", array[count]);
```

也可以进一步简化该 for 循环，如下所示（如果您无法理解对测试表达式所做的这种修改，请复习第 4 天的课程）：

```
for (count = 0; count < 1000 && array[count]; )
    printf("%d", array[count++]);
```

也可以在 for 语句后面跟一条空语句，让所有的工作都由 for 语句本身来完成。请记住，空语句是一个单独占一行的分号。例如，要将包含 1000 个元素的数组中的每个元素初始化为 50，可以这样编写代码：

```
for (count = 0; count < 1000; array[count++] = 50)
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
    ;
```

在上述 for 语句中，将 50 赋给数组每个元素的工作是由递增部分完成的。一种更好的方法是将该语句编写为：

```
for (count = 0; count < 1000; array[count++] = 50)
{
    ;
}
```

将分号放在语句块（两个花括号）中，可以更清楚地看出 for 语句体没有执行任何工作。

第 4 天的课程中指出过，逗号运算符最常被用于 for 语句中。可以使用逗号将两个子表达式组合成一个表达式，从左到右计算两个子表达式，而整个表达式的值为右边的子表达式的值。通过使用逗号运算符，可以让 for 语句的每个部分都完成多项任务。

假设有两个各包含 1000 个元素的数组：a[]和 b[]，如果您想将 a[]的内容反向复制到 b[]中，即使得 b[0] = a[999]，b[1] = a[998]，等等，则可以使用下面的 for 语句实现：

```
for (i = 0, j = 999; i < 1000; i++, j--)
    b[j] = a[i];
```

逗号运算符被用来初始化两个变量 i 和 j，还被用来在每次循环中对这两个变量进行递增。

for 语句的语法如下：

```
for (initial; condition; increment)
    statement(s)
```

其中 initial 可以是任何合法的表达式，通常是一个将变量设置为特定值的赋值语句。

condition 可以是任何合法的表达式，通常是一个关系表达式。当 condition 为假（0）时，for 语句将终止，然后接着执行 statement(s)后面的第一条语句；否则将执行 statement(s)中的语句。

increment 可以是任何合法的表达式，通常是一个将初始化表达式设置的变量递增的表达式。

statement(s)是只要条件为真便执行的语句。

for 语句是一种循环语句，可以包含初始化、测试条件和递增部分。for 语句首先执行初始化表达式，然后检查条件。如果条件为真，则执行其中的语句，再执行递增表达式。然后重新检查条件，并不断执行循环，直到条件为假为止。

下面是三个 for 语句范例：

范例 1：

```
/*Prints the value of x as it counts from 0 to 9*/
int x;
for (x = 0; x <10; x++)
    printf( "\nThe value of x is %d", x );
```

范例 2：

```
/*Obtains values from the user until 99 is entered */
int nbr = 0;
for ( ; nbr != 99; )
    scanf( "%d", &nbr );
```

范例 3：

```
/* Lets user enter up to 10 integer values        */
/* Values are stored in an array named value. If 99 is */
/* entered, the loop stops                                */
int value[10];
int ctr,nbr=0;
for (ctr = 0; ctr < 10 && nbr != 99; ctr++)
{
    puts("Enter a number, 99 to quit ");
    scanf("%d", &nbr);
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
    value[ctr] = nbr;
}
```

6.2.2　嵌套 for 语句

for 语句中可以包含另一条 for 语句，这叫做嵌套（第 4 天的课程介绍了 if 语句的嵌套）。通过对 for 语句进行嵌套，可以完成一些复杂的编程。程序清单 6.2 虽然不是一个复杂的程序，但它演示了如何嵌套 for 语句。

程序清单 6.2　　nestfor.c：嵌套 for 语句

```
1:  /* Demonstrates nesting two for statements */
2:
3:  #include <stdio.h>
4:
5:  void draw_box( int, int);
6:
7:  int main( void )
8:  {
9:      draw_box( 8, 35 );
10:
11:     return 0;
12: }
13:
14: void draw_box( int row, int column )
15: {
16:     int col;
17:     for ( ; row > 0; row--)
18:     {
19:         for (col = column; col > 0; col--)
20:             printf("X");
21:
22:         printf("\n");
23:     }
24: }
```

该程序的输出如下：

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

分析：该程序的主要工作是由第 20 行完成的。运行该程序时，将在屏幕上打印 280 个“X”，它们被分成 8 行，每行 35 个。该程序只包含一个打印 X 的命令，但被嵌套在两个循环中。

在该程序清单中，第 5 行是 draw_box()的函数原型，该函数接受两个 int 变量 row 和 colum，它们分别是要打印的 X 的行数和列数。在第 9 行，main()函数调用 draw_box()，并将参数 8 和 35 传递给它。

仔细查看 draw_box()，您发现有两项内容不能理解。首先，为何要声明局部变量 col，其次，为何要包含第 22 行的 printf()。通过查看两个 for 循环，您将明白这两点。

从第 17 行开始是第一个 for 循环。其中省略了初始化部分，因为已经将 row 的初始值传递给函数。从条

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

件可知，该 for 循环将一直执行，直到 row 为 0。首次执行到第 17 行时，row 的值为 8；因此程序将继续执行到第 19 行。

从第 19 行开始是第二条 for 语句，该行将传递过来的参数 column 的值复制给类型为 int 的局部变量 col。col 的初始值为 35（通过 column 传递的值），而 column 的值保持不变。由于 col 大于 0，因此第 20 行被执行，它打印一个 X。然后 col 的值被减 1，并继续循环。当 col 的值为 0 后，该 for 循环将结束，然后执行第 22 行。第 22 行导致打印在新的一行进行（有关打印，将在第 7 天的课程中介绍）。移到屏幕的下一行后，程序便执行到了第一个 for 循环的语句的结尾，因此接着执行递增表达式。该表达式将 row 的值减 1，使之等于 7。这样将继续执行第 19 行。注意，col 最后一次被使用时，值为 0。如果使用 column，而不是 col，则第二次执行第一个 for 循环时，条件将为假，因为 column 永远不会大于 0。因此，程序只打印第一行。请删除第 19 行的初始化表达式，并将两个 col 替换为 column，看看实际发生的情况。

应　该	不应该
在 for 循环中使用空语句时，别遗漏了分号，请让分号占位符单独占一行，或将它放在 for 语句的后面。让分号单独占一行，代码将更为清晰，如下所示： `for (count = 0; count < 1000;` `array[count] = 50) ;` `/* note space! */`	不要在 for 语句中执行过多的处理工作。虽然可以使用逗号分隔符，但把一些处理工作放到循环体中，代码将更清晰。

6.2.3 while 语句

while 语句也叫 while 循环，它不断地执行一个语句块，直到条件为假为止。while 语句的格式如下：

```
while (condition)
    statement
```

其中 condition 可以是任何表达式，statement 是一条语句或一个复合语句。程序执行到 while 语句后，将发生以下事件：

1. 计算表达式 condition；
2. 如果 condition 为假（即为零），while 语句将结束，然后执行 statement 后面的第一条语句。
3. 如果 condition 为真（即非零），则执行 statement 中的语句。
4. 返回到第 1 步。

while 语句的工作原理如图 6.3 所示。

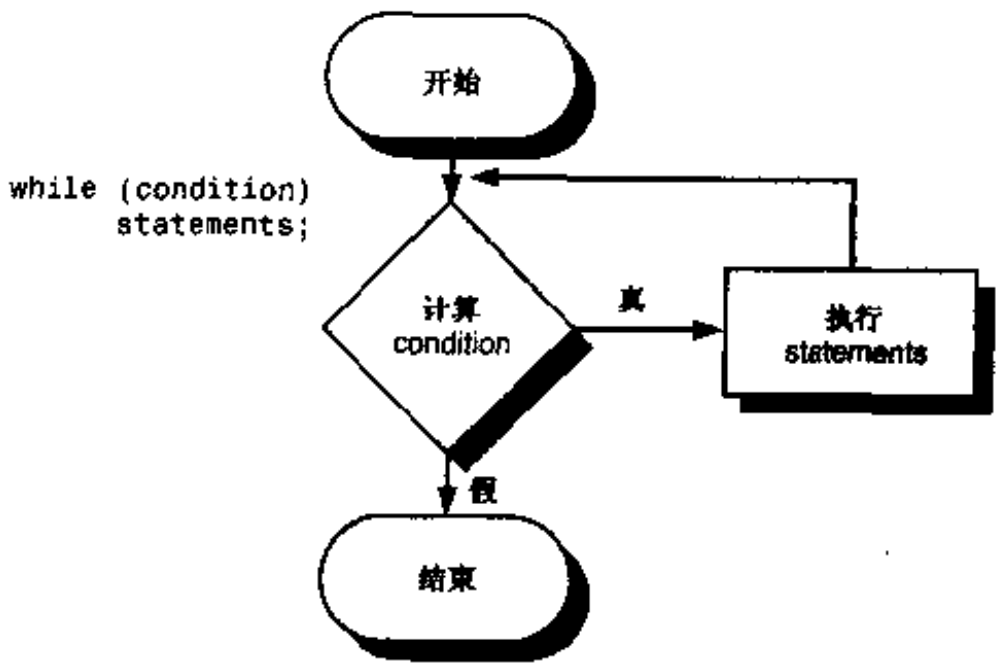


图 6.3　while 语句的工作原理

程序清单 6.3 是一个简单的程序，它使用一条 while 语句来打印数字 1～20（这与程序清单 6.1 中的 for 语句完成的任务相同）。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 6.3　　whileat.c：一条简单的 while 语句

```
1:  /* Demonstrates a simple while statement */
2:
3:  #include <stdio.h>
4:
5:  int count;
6:
7:  int main( void )
8:  {
9:      /* Print the numbers 1 through 20 */
10:
11:     count = 1;
12:
13:     while (count <= 20)
14:     {
15:         printf("%d\n", count);
16:         count++;
17:     }
18: return 0;
19: }
```

该程序的输出如下：

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

分析：现在分析程序清单 6.3，并将它同程序清单 6.1 进行比较，后者使用 for 语句来执行相同的任务。第 11 行将 count 的值初始化为 1。由于 while 语句不包含初始化部分，因此必须在执行 while 循环之前对变量进行初始化。第 13 行是 while 语句，其中包含的条件语句（cont <= 20）与程序清单 6.1 相同。在 while 循环中，第 16 行负责将 count 的值递增。如果省略第 16 行，情况将如何呢？循环将不断执行下去，因为 count 的值总为 1，即总是小于 20。

您可能注意到了，while 语句实际上就是一条没有初始化部分和递增部分的 for 语句，因此下面的语句：

```
for ( ; condition ; )
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

与下述语句等价：

```
while (condition)
```

由于这种等价性，使用 for 语句能够完成的任何工作都可以使用 while 语句来完成。使用 while 语句时，必须首先使用单独的语句来完成所有必要的初始化工作，同时必须在 while 循环体内使用一条语句来完成递增工作。

当必须执行初始化和递增工作时，大多数经验丰富的程序员都喜欢使用 for 语句，而不是 while 语句。这种偏好主要基于源代码的可读性。使用 for 语句时，初始化、测试和递增表达式在同一个地方，因此易于找到并修改它们。使用 while 语句时，初始化表达式和递增表达式位于不同的地方，可能不那么明显。

while 语句的语法如下：

```
while (condition)
   statement(s)
```

其中 condition 可以是任何合法的表达式，通常为一个关系表达式。当 condition 为假（即零）时，while 语句将结束，然后执行 statement(s)后面的第一条语句；否则执行 statement(s)中的语句。

statement(s)是只要 condition 为真便执行的语句。

while 语句是一种循环语句，它让您能够重复执行一条语句或一个语句块，只要条件为真（即非零）。当首次执行到 while 语句时，如果条件为假，则 statement(s)一次也不会执行。

下面是三个 while 语句的范例：

范例 1：

```
int x = 0;
while (x < 10)
{
    printf("\nThe value of x is %d", x );
    x++;
}
```

范例 2：

```
/* get numbers until you get one greater than 99 */
int nbr=0;
while (nbr <= 99)
   scanf("%d", &nbr );
```

范例 3：

```
/* Lets user enter up to 10 integer values      */
/* Values are stored in an array named value. If 99 is */
/* entered, the loop stops                              */
int value[10];
int ctr = 0;
int nbr;
while (ctr < 10 && nbr != 99)
{
    puts("Enter a number, 99 to quit ");
    scanf("%d", &nbr);
    value[ctr] = nbr;
    ctr++;
}
```

6.2.4 嵌套 while 语句

与 for 语句和 if 语句一样，while 语句也可以嵌套。程序清单 6.4 是一个嵌套 while 语句的范例。虽然这不是使用 while 语句的最佳方式，但它说明了一些新思想。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 6.4　　whiles.c: 嵌套 while 语句

```
1:  /* Demonstrates nested while statements */
2:
3:  #include <stdio.h>
4:
5:  int array[5];
6:
7:  int main( void )
8:  {
9:     int ctr = 0,
10:        nbr = 0;
11:
12:    printf("This program prompts you to enter 5 numbers\n");
13:    printf("Each number should be from 1 to 10\n");
14:
15:    while ( ctr < 5 )
16:    {
17:       nbr = 0;
18:       while (nbr < 1 || nbr > 10)
19:       {
20:          printf("\nEnter number %d of 5: ", ctr + 1 );
21:          scanf("%d", &nbr );
22:       }
23:
24:       array[ctr] = nbr;
25:       ctr++;
26:    }
27:
28:    for (ctr = 0; ctr < 5; ctr++)
29:       printf("Value %d is %d\n", ctr + 1, array[ctr] );
30:
31:    return 0;
32: }
```

该程序的运行情况如下：

```
This program prompts you to enter 5 numbers
Each number should be from 1 to 10

Enter number 1 of 5: 3

Enter number 2 of 5: 6

Enter number 3 of 5: 3

Enter number 4 of 5: 9

Enter number 5 of 5: 2

Value 1 is 3
Value 2 is 6
Value 3 is 3
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
Value 4 is 9
Value 5 is 2
```

分析：与前面的程序清单一样，该程序清单的第一行也是注释，它描述了程序的功能；而第 3 行是一条 #include 语句，用于包含标准输入/输出头文件。第 5 行声明了一个数组（名为 array），该数组可以存储 5 个整型值。函数 main()包含另外两个局部变量 ctr 和 nbr（第 9 和 10 行）。这些变量在声明时被初始化为 0，另外第 9 行的结尾将逗号运算符用作分隔符，这使得将 nbr 声明为 int 类型时，无需再指定类型。很多 C 语言程序员常以这种方式来声明变量。第 12 和 13 行打印消息，指出该程序的功能以及用户应输入的内容。第 15～26 行是第一个 while 命令及其语句；第 18～22 行是被嵌套的 while 循环及其语句，它们都是第一个 while 语句的组成部分。

外面的循环将不断执行，直到 ctr 大于或等于 5（第 15 行）。当 ctr 小于 5 时，第 17 行将 nbr 设置为 0，第 18～22 行（被嵌套的 while 语句）将用户输入的数字赋给 nbr，第 24 行将这个数字放到数组 array 中，第 25 行将 ctr 的值加 1。然后，重新开始循环。因此外面的循环获取 5 个数字，并将它们放到数组 array 中，并使用 ctr 为索引。

内面的循环是一种使用 while 语句的好方式。只有数字 1～10 是合法的，因此在用户输入合法的数字之前，程序不会向前执行，这是由第 18～22 行实现的。该 while 语句的意思是，如果用户输入的数字小于 1 或大于 10，程序将打印一条消息，提示用户输入一个数字，然后读取用户输入的数字。

第 28 和 29 行打印存储在数组 array 中的值。由于 while 语句是使用变量 ctr 完成的，因此 for 语句可以重用该变量。for 语句将 ctr 的值初始化为 0，然后循环 5 次，每次将 ctr 的值加 1，并打印 ctr 加 1 的值（因此计数是从零开始的）以及数组中相应的值。

做为练习，您可以修改这个程序的两个地方。首先可以让程序接受 1～100 之间的值，面不是 1～10 之间的值；还可以修改程序接受的值的数目。当前，该程序接受 5 个值，您可以尝试将其修改为接受 10 个值。

应　读	不应读
如果需要在循环中执行初始化和递增操作，应使用 for 语句，而不是 while 语句。for 语句将初始化、条件和递增语句放在一起，而 while 语句不是。	不要使用下面的方式： while (x) 而应使用下面的方式： while (x != 0) 虽然这两种方式都管用，但当您调试程序（查找代码中的错误）时，后者更为清晰。在编译时，这两种方式生成的代码几乎完全相同。

6.2.5 do...while 循环

C 语言中的第三种循环结构是 do...while 循环，它在指定的条件为真时不断执行一个语句块。do...while 循环在每次循环结束后检测条件，而不像 for 循环和 while 循环那样，在开始前检测条件。

do...while 循环的结构如下：

```
do
        statement
while (condition);
```

其中 condition 可以是任何合法的表达式，而 statement 是一条语句或一个复合语句。程序执行到 do...while 语句后，将发生以下事件：

1. 执行 statement 中的语句。
2. 计算 condition。如果为真，则返回到第 1 步；否则循环结束。

do...while 循环的工作原理如图 6.4 所示。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

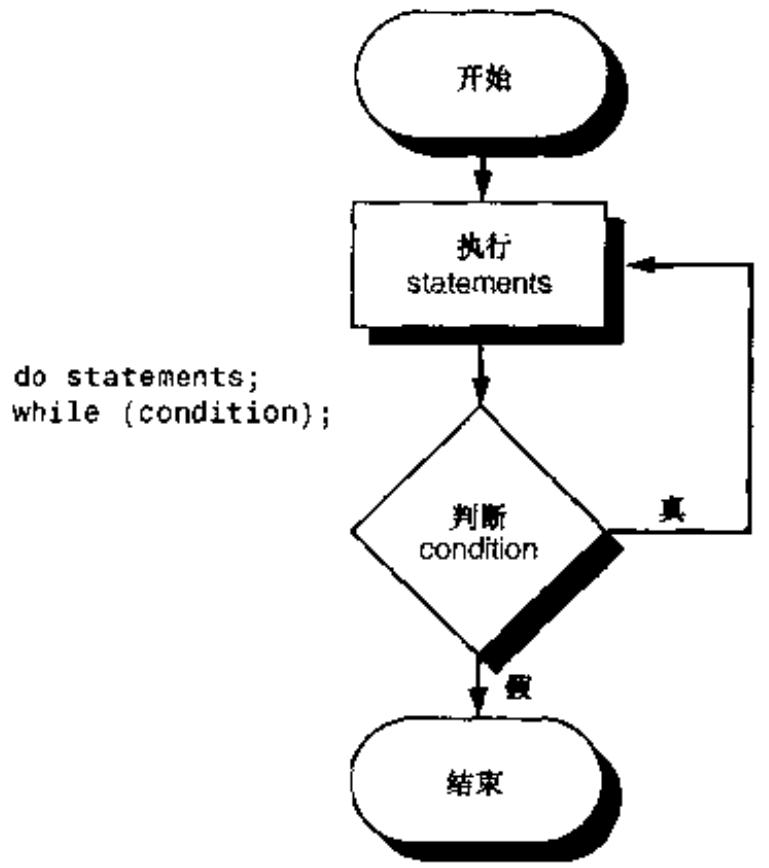


图 6.4　do...while 循环的工作原理

do...while 循环中的语句至少执行一次，这是因为条件是在每次循环结束后（而不是开始前）检测的；而 for 循环和 while 循环在循环前检测条件，因此如果其中的条件开始就为假，则循环中的语句一次也不执行。

do...while 循环用得没有 while 和 for 循环那么多，它最适合循环中的语句至少必须执行一次的情况。当然，您也可以使用 while 循环来完成这样的工作，方法是确保程序首次执行到该循环时条件为真，但使用 do...while 循环更为简单明了。

程序清单 6.5 是一个 do...while 循环的例子。

程序清单 6.5　　do.c：一个简单的 do..while 循环

```
1:  /* Demonstrates a simple do...while statement */
2:
3:  #include <stdio.h>
4:
5:  int get_menu_choice( void );
6:
7:  int main( void )
8:  {
9:     int choice;
10:
11:    choice = get_menu_choice();
12:
13:    printf("You chose Menu Option %d\n", choice );
14:
15:    return 0;
16: }
17:
18: int get_menu_choice( void )
19: {
20:    int selection = 0;
21:
22:    do
23:    {
24:       printf("\n" );
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
25:         printf("\n1 - Add a Record" );
26:         printf("\n2 - Change a record");
27:         printf("\n3 - Delete a record");
28:         printf("\n4 - Quit");
29:         printf("\n" );
30:         printf("\nEnter a selection: " );
31:
32:         scanf("%d", &selection );
33:
34:      }while ( selection < 1 || selection > 4 );
35:
36:      return selection;
37:  }
```

该程序的运行情况如下：

```
1 - Add a Record
2 - Change a record
3 - Delete a record
4 - Quit

Enter a selection: 8
1 - Add a Record
2 - Change a record
3 - Delete a record
4 - Quit

Enter a selection: 4
You chose Menu Option 4
```

分析：该程序提供了一个包含4个选项的菜单。用户可以选择其中的一个选项，然后程序将打印用户选择的数字。本书后面的程序将运用并扩展这一概念。现在，您应该能够理解该程序清单中的大部分内容。main()函数（第7～16行）中的内容都是以前介绍过的。



注意：main()函数的函数体可以书写成一行，如下所示：

```
printf( "You chose Menu Option %d", get_menu_option() );
```

如果您要扩展该程序，并根据选择做出相应的操作，则需要get_menu_choice()返回的值，因此将这个值赋给一个变量（如choice）是明智的。

第18～37行为get_menu_choice()函数。该函数在屏幕上显示菜单（第24～30行），并读取用户的选择。由于您至少需要显示一次菜单，以读取用户的选择，因此使用do...while循环是合适的。就这个程序而言，将不断地显示菜单，直到用户做出了有效的选择。do...while语句的while部分位于第34行，它验证选择的值（被命名为selection）是否有效。如果用户输入的值不在1～4之间，将再次显示菜单，并提示用户输入新的值。用户输入有效的值后，程序将接着执行第36行，即返回变量selection的值。

while语句的语法如下：

```
do
{
        statement(s)
}while (condition);
```

其中condition可以是任何合法的表达式，通常为一个关系表达式。当condition为假（即零）时，do...while循环将结束，然后接着执行while语句后面的第一条语句；否则返回到do语句，并执行statement(s)中的语句。

statement(s)可以是单条语句，也可以是一个语句块。当程序首次执行到该循环时，这些语句将执行，然

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

后只要 condition 为真，这些语句仍将执行。

do...while 语句是一种循环语句，它让您能够在条件为真的情况下，不断地执行一条语句或一个语句块。与 while 语句不同，do...while 循环体中的语句至少执行一次。

下面是三个 do...while 循环的例子：

范例 1：

```
/* prints even though condition fails! */
int x = 10;
do
{
    printf("\nThe value of x is %d", x );
}while (x != 10);
```

范例 2：

```
/* gets numbers until the number is greater than 99 */
int nbr;
do
{
   scanf("%d", &nbr );
}while (nbr <= 99);
```

范例 3：

```
/* Enables user to enter up to 10 integer values        */
/* Values are stored in an array named value. If 99 is  */
/* entered, the loop stops                              */
int value[10];
int ctr = 0;
int nbr;
do
{
        puts("Enter a number, 99 to quit ");
        scanf( "%d", &nbr);
        value[ctr] = nbr;
        ctr++;
}while (ctr < 10 && nbr != 99);
```

6.3　嵌套循环

术语嵌套循环指的是一个循环包含在另一个循环中。您已经见过一些嵌套语句的范例。C 语言对于对循环进行嵌套没有任何限制，只是每个内部循环必须完全位于外部循环中，而不能相互交叠。因此下面的代码是非法的：

```
for ( count = 1; count < 100; count++)
{
    do
    {
            /* the do...while loop */
} /* end of for loop */
     }while (x != 0);
```

如果将 do...while 循环完全放在 for 循环中，则没有任何问题：

```
for (count = 1; count < 100; count++)
{
    do
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
    {
        /* the do...while lcop */
    }while (x != 0);
} /* end of for loop */
```

使用嵌套循环时，请记住，修改内部循环也可能影响外部循环。然而，内部循环可能独立于外部循环中的所有变量，在这种情况下，修改内部循环不会影响外部循环。在前一个例子中，如果内部的 do...while 循环修改了变量 count 的值，将影响外部的 for 循环的执行次数。

良好的缩进风格可提高嵌套循环的可读性。每一级循环都应相对于前一级循环进行缩进，这样每一级循环中的代码便一目了然。

应　读	不应读
当您知道循环至少应执行一次时，应使用 do...while 循环。	不要让循环交叠。可以嵌套循环，但一个循环必须完全在另一个循环内。

6.4 总　结

学习今天的课程后，您便可以开始编写真正的 C 程序了。

C 语言中有三种用于控制程序执行的循环语句：for、while 和 do...while。这三种循环都让程序能够根据特定变量的情况，执行一个语句块零次、一次或多次。很多编程任务都可以通过使用这些循环语句、重复执行代码来完成。

虽然这三种语句可用于完成相同的任务，但其中每一种又各不相同。for 语句能够在一个命令中包含初始化、检测和递增等三个部分；while 语句只要条件为真便执行循环；而 do...while 语句至少执行其语句一次，并在条件为真时继续不断执行。

嵌套指的是将一个命令放在另一个命令中。C 语言允许嵌套任何命令。第 4 天的课程介绍了如何嵌套 if 语句，而今天的课程介绍了如何嵌套 for、while 和 do...while 语句。

6.5 问与答

问：如何确定应使用哪种程序控制语句——for 语句、while 语句还是 do...while 语句？

答：从今天课程中介绍的语法格式可知，这三种语句都可用于解决循环问题；但每种语句独有其最适合的情况。在循环中需要完成初始化和递增工作时，for 语句最合适；当知道需要满足的条件，且预先无法知道循环的次数时，while 最适合；当一组语句至少必须执行一次时，do...while 最合适。由于这三种语句都可用于解决大部分问题，因此最好三种语句都学习，然后对编程情形进行评估，以确定哪种语句最适合。

问：循环最多可嵌套多少层？

答：可以嵌套任意层。如果在程序中需要嵌套两层以上的循环，则应考虑使用函数。在这种情况下，很难辨别所有的花括号，此时使用函数将使代码更容易理解。

问：可以嵌套不同的循环命令吗？

答：可以嵌套 if、for、while、do...while 或其他任何命令。您将发现，您编写的很多程序需要嵌套这些命令。

6.6 作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

6.6.1　小测验

1. 数组的第一个索引值是多少？
2. for 语句和 while 语句之间的区别是什么？
3. while 语句和 do...while 语句之间的区别是什么？
4. for 语句能够完成的工作，while 语句也能完成，这话是否正确？
5. 嵌套语句时，必须切记的一点是什么？
6. 可以将 while 语句嵌套到 do...while 语句中吗？
7. for 语句由哪 4 部分组成？
8. while 语句由哪两部分组成？
9. do...while 语句由哪两部分组成？

6.6.2　练习

1. 声明一个存储 50 个 long 值的数组。
2. 编写一条语句，将 123.456 赋给练习 1 中的数组的第 50 个元素。
3. 下面的语句执行完毕后，x 的值为多少？

```
for (x = 0; x < 100, x++) ;
```

4. 下面的语句执行完毕后，ctr 的值为多少？

```
for (ctr = 2; ctr < 10; ctr += 3) ;
```

5. 下面的代码打印多少个“X”？

```
for (x = 0; x < 10; x++)
        for (y = 5; y > 0; y--)
            puts("X");
```

6. 编写一条 for 语句，从 1 数到 100，每次的间隔为 3 秒。
7. 编写一条 while 语句，从 1 数到 100，每次的间隔为 3 秒。
8. 编写一条 do...while 语句，从 1 数到 100，每次的间隔为 3 秒。
9. 排错：下面的代码片段有什么错误？

```
record = 0;
while (record < 100)
{
        printf( "\nRecord %d ", record );
        printf( "\nGetting next number..." );
}
```

10. 排错：下面的代码片段有什么错误（MAXVLUES 没有问题）？

```
for (counter = 1; counter < MAXVALUES; counter++);
        printf("\nCounter = %d", counter );
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第 7 天课程　信息读写基础

在大部分程序中，都需要在屏幕上显示信息或从键盘读取信息。前面课程中的很多程序都执行了这样的任务，但您可能还不知道这是如何完成的。今天将学习以下内容：

- 输入/输出语句的基本知识；
- 如何使用库函数 printf()和 puts()将信息显示到屏幕上；
- 如何格式化被显示到屏幕上的信息；
- 如何使用库函数 scanf()从键盘读取数据；

今天的课程并不会介绍有关这些主题的所有知识，但其中提供的信息对于您开始编写真正的程序而言足够了。本书后面的课程将更详细地介绍这些主题。

7.1　在屏幕上显示信息

大多数程序都需要在屏幕上显示信息，两种最常见的显示信息的方式是使用库函数 printf()和 puts()。

7.1.1　printf()函数

printf()函数位于标准 C 语言库中，也是 ANSI 标准的组成部分，它是最通用的、在屏幕上显示信息的方式。本书前面的很多程序都使用了 printf()函数，下面介绍该函数的工作原理。

在屏幕上打印文本消息很简单，只须调用 printf()函数，将需要显示的消息用双引号括起，并传递给该函数即可。例如，要在屏幕上显示 How Now Brown Cow!，可以这样编写代码：

```
printf("How Now Brown Cow!");
```

除了文本消息外，您还经常需要显示变量的值，这比显示消息更复杂些。例如，假设您要将数值变量 myNumber 的值以及一些标识文本显示到新一行的开始位置，可以这样使用 printf()函数：

```
printf("\nThe value of myNumber is %d", myNumber);
```

如果 myNumber 的值为 12，则显示结果如下：

```
The value of myNumber is 12
```

在上述范例中，给 printf()传递了两个参数。第一个参数叫格式化字符串，被包含在双引号中；第二个参数是包含要打印的值的变量名（myNumber）。

7.1.2　格式化字符串

格式化字符串指定如何格式化输出，由以下三部分组成：

- 字面文本：按输入方式显示。在前一个例子中，格式化字符串为从 T 到%（不包括%）之间的字符。
- 转义序列：提供特殊的格式化控制，由一个反斜杠和一个字符组成。在前面的范例中，\n 为转义序列，它被称为换行符，意思是“移到下一行的开始位置”。转义序列也可用于打印特定的字符，表 7.1 列出了一些常用的转义序列。
- 转换说明符：由百分符（%）和一个字符组成。在前面的范例中，%d 为转换说明符。转换说明符告

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

诉 printf()函数如何解释要打印的变量。%d 告诉 printf()将变量 myNumber 视为有符号的十进制整数。

表 7.1　　最常用的转义序列

序　列	含　义
\a	振铃
\b	退格
\f	换页
\n	换行
\r	回车
\t	水平制表符
\v	垂直制表符
\\	反斜杠
\?	问号
\'	单引号
\"	双引号

7.1.3　转义序列

转义序列用于通过移动光标来控制输出的位置，也可用于打印那些本来有特殊含义的字符。例如，要打印一个反斜杠，可以在格式化字符串中包含两个反斜杠。第一个反斜杠告诉 printf()，将第二个反斜杠解释为字面字符，而不是转义序列的开始标记。通常，反斜杠告诉 printf()，以特殊的方式解释下一个字符，下面是一些例子：

- n：字符 n；
- \n：换行符；
- \"：双引号；
- "：字符串的开始或结尾。

表 7.1 列出了 C 语言中最常用的转义序列；程序清单 7.1 演示了一些最常用的转义序列。

程序清单 7.1　　escape.c：使用转义序列

```
1:  /* Demonstration of frequently used escape sequences */
2:
3:  #include <stdio.h>
4:
5:  #define QUIT  3
6:
7:  int  get_menu_choice( void );
8:  void print_report( void );
9:
10: int main( void )
11: {
12:     int choice = 0;
13:
14:     while (choice != QUIT)
15:     {
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
16:        choice = get_menu_choice();
17:
18:        if (choice == 1)
19:           printf("\nBeeping the computer\a\a\a" );
20:        else
21:        {
22:           if (choice == 2)
23:              print_report();
24:        }
25:     }
26:     printf("You chose to quit!\n");
27:
28:     return 0;
29: }
30:
31: int get_menu_choice( void )
32: {
33:     int selection = 0;
34:
35:     do
36:     {
37:        printf( "\n" );
38:        printf( "\n1 - Beep Computer" );
39:        printf( "\n2 - Display Report");
40:        printf( "\n3 - Quit");
41:        printf( "\n" );
42:        printf( "\nEnter a selection:" );
43:
44:        scanf( "%d", &selection );
45:
46:     }while ( selection < 1 || selection > 3 );
47:
48:     return selection;
49: }
50:
51: void print_report( void )
52: {
53:     printf( "\nSAMPLE REPORT" );
54:     printf( "\n\nSequence\tMeaning" );
55:     printf( "\n========\t=======" );
56:     printf( "\n\\a\t\tbell (alert)" );
57:     printf( "\n\\b\t\tbackspace" );
58:     printf( "\n...\t\t...");
59: }
```

该程序的运行情况如下：

```
1 - Beep Computer
2 - Display Report
3 - Quit

Enter a selection:1
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
Beeping the computer

1 - Beep Computer
2 - Display Report
3 - Quit

Enter a selection:2

SAMPLE REPORT
Sequence        Meaning
=========       =======
\a              bell (alert)
\b              backspace
...             ...
1 - Beep Computer
2 - Display Report
3 - Quit

Enter a selection:3
You chose to quit!
```

分析：与前面的范例相比，程序清单 7.1 更长，但包含了一些需要注意的内容。第 3 行包含了头文件 stdio.h，因为该程序需要使用 printf()。第 5 行定义了一个名为 QUIT 的常量。第 3 天的课程介绍过，#define 使得使用常量 QUIT 与使用 3 等价。第 7 和 8 行是函数原型。该程序有两个函数：get_menu_choice()和 print_report()。get_menu_choice()是在第 31～49 行定义的，它与程序清单 6.5 中的菜单函数类似。第 37 和 41 行调用 printf()来换行。第 38、39、40 和 42 行打印文本，它们也使用了转义字符换行符。可以将 38 行修改为如下所示，以删除第 37 行：

```
printf( "\n\n1 - Beep Computer" );
```

然而，留下第 37 行可提高程序的可读性。

在 main()函数中，从第 14 行开始一个 while 循环，该循环不断执行，直到 choice 为 QUIT 为止。由于 QUIT 是一个常量，因此可将其替换为 3，但如果这样做，程序将没有那么清晰。第 16 行读取用户的选择，并将其存储在变量 choice 中，然后第 18～25 行的 if 语句对该变量进行分析。如果用户选择 1，则执行第 19 行——在新的一行中打印一条消息，并振铃三次；如果用户选择 2，则执行第 23 行——调用函数 print_report()。

print_report()函数是在第 51～59 行定义的，该函数很简单，它表明使用 printf()和转义序列在屏幕上打印信息很容易。第 54～58 行使用了换行符和制表符转义序列\t。制表符用于将报告的各列垂直对齐。乍一看，第 56 和 57 行令人迷惑，但只要仔细的从左向右阅读，便能理解其含义。第 56 行首先在屏幕上换行，然后打印\a 和两个制表符（\t\t），最后是一些描述性文本（bell (alert)）。第 57 行的格式与此相同。

该程序打印报告题目、列标题以及表 7.1 的前两行。本章最后的练习 9 要求您完成该程序，使之打印表 7.1 的其他内容。

转换说明符

对于要打印的每个变量，格式化字符串中都必须包含一个相应的转换说明符，这样 printf()将根据转换说明符打印每个变量。有关这方面的内容将在第 15 天的课程中做更详细的介绍。就现在而言，您只须使用与要打印的变量的类型对应的转换说明符即可。

这是什么意思呢？如果要打印一个有符号的十进制整数（类型为 int 和 long），则使用转换说明符%d；对于无符号的十进制整数（类型为 unsigned int 和 unsigned long），则使用%u；对于浮点变量（类型为 float 和

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

double），则使用%f。最常用的转换说明符如表 7.2 所示。

表 7.2　　最常用的转换说明符

说 明 符	含　义	要转换的类型
%c	单个字符	char
%d	有符号的十进制整数	int、short
%ld	有符号的十进制长整数	long
%f	十进制浮点数	float、double
%s	字符串	char 数组
%u	无符号的十进制整数	unsigned int、unsigned short
%lu	无符号的十进制长整数	Unsigned long



注意：使用 printf()的程序必须包含头文件 stdio.h。

格式说明符中的字面文本可以是除转义序列和转换说明符之外的任何东西，它将逐字被打印（包括其中的空格）。

要打印多个变量的值，该如何办呢？一条 printf()语句可以打印任意数目的变量，但对于每一个变量，格式化字符串中都必须包含一个相应的转换说明符。转换说明符和变量必须成对出现，前者位于左边，而后者位于右边。在下面的语句中：

```
printf("Rate = %f, amount = %d", rate, amount);
```

变量 rate 和说明%f 符是一对；而变量 amount 和说明符%d 是一对。转换说明符在格式化字符串中的位置是变量的输出位置。如果传递给 printf()的变量数目多于转换说明符的数目，则没有相应说明符的变量将不被打印。如果说明符的数目多于变量数，则没有相应变量的说明符将打印“垃圾值”。

使用 printf()打印变量的值是不受限制的。参数可以是任何有效的表达式，例如，要打印 x 与 y 的和，可以这样编写代码：

```
total = x + y;
printf("%d", total);
```

也可以这样编写代码：

```
printf("%d", x + y);
```

程序清单 7.2 演示了如何使用 printf()。第 15 天的课程将更详细地介绍 printf()。

程序清单 7.2　　nums.c：使用 printf()显示数值

```
1:  /* Demonstration using printf() to display numerical values. */
2:
3:  #include <stdio.h>
4:
5:  int a = 2, b = 10, c = 50;
6:  float f = 1.05, g = 25.5, h = -0.1;
7:
8:  int main( void )
9:  {
10:     printf("\nDecimal values without tabs: %d %d %d", a, b, c);
11:     printf("\nDecimal values with tabs: \t%d \t%d \t%d", a, b, c);
12:
13:     printf("\nThree floats on 1 line: \t%f\t%f\t%f", f, g, h);
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
14:     printf("\nThree floats on 3 lines: \n\t%f\n\t%f\n\t%f", f, g, h);
15:
16:     printf("\nThe rate is %f%%", f);
17:     printf("\nThe result of %f/%f = %f\n", g, f, g / f);
18:
19:     return 0;
20: }
```

该程序的输出如下：

```
Decimal values without tabs: 2 10 50
Decimal values with tabs:       2       10      50
Three floats on 1 line:         1.050000        25.500000       0.100000
Three floats on 3 lines:
        1.050000
        25.500000
        -0.100000
The rate is 1.050000%
The result of 25.500000/1.050000 = 24.285715
```

分析：程序清单 7.2 打印 6 行信息。第 10 和 11 行分别打印三个十进制数：a、b 和 c。第 10 行打印它们时没有添加制表符，而第 11 行添加了。第 13 和 14 行分别打印三个浮点变量 f、g 和 h。其中第 13 行在一行中打印它们，而第 14 行则在三行中打印它们。第 16 行打印浮点变量 f 和一个百分号。由于百分号通常标记着要打印一个变量，因此要打印一个百分号，必须使用两个百分号，这类似于转义字符反斜杠。第 17 行说明了最后一个概念——使用转换说明符打印值时不一定非得要使用变量，也可以使用表达式，如 g / f，甚至可以使用常量。

应　该	不 应 该
在不同的 printf()语句中打印多行信息时，一定要使用转义字符换行符。	不要在一条 printf()语句中打印多行文本。在大多数情况下，使用多条打印语句来打印多行信息，比在一条打印语句中使用多个换行符（\n）来打印它们更清晰。 不要拼错了头文件 stdio.h。很多 C 语言程序员将其错误的拼写为 studio.h。

printf()函数的语法如下：

```
#include <stdio.h>
printf( format-string[,arguments,...]);
```

函数 printf()接受一系列的参数，其中每个参数应于格式化字符串中的一个转换说明符。printf()将格式化后的信息打印到标准输出设备（通常为显示器屏幕）上。使用 printf()时，应包含标准输入/输出头文件 STDIO.H。

format-string 是必不可少的，但参数是可选的。每一个参数都必须有一个对应的转换说明符。表 7.2 列出了一些最常用的转换说明符。

format-string 也可以包含转义序列，表 7.1 列出了一些最常用的转义序列。

下面是两个调用 printf()函数的范例及其输出：

范例 1：

```
#include <stdio.h>
int main( void )
{
    printf("This is an example of something printed!");
    return 0;
}
```

该范例的输出如下：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
This is an example of something printed!
```

范例 2：

```
printf("This prints a character, %c\na number, %d\na floating \
point, %f", 'z', 123, 456, 789)
```

该范例的输出如下：

```
This prints a character, z
a number, 123
a floating point, 456.789
```



提示：您可能注意到了，第二个范例中的 printf()函数占了两行。在第一行的末尾使用了一个反斜杠（\）指出字符串将延续到下一行，因此编译器将把这两行代码视为一行。

7.1.4 使用 puts()显示消息

函数 puts()也可用来在屏幕上显示文本消息，但它不能显示数值变量。puts()函数接受一个字符串参数，显示该参数并自动换行。例如语句：

```
puts("Hello, world.");
```

与下面的语句等效：

```
printf("Hello, world.\n");
```

传递给 puts()的字符串中可以包含转义序列（包括\n），效果与用于 printf()函数中相同（有关最常用的转义序列，请参阅表 7.1）。

和 printf()一样，使用 puts()的程序也必须包含头文件 stdio.h。但需要注意的是，在程序中只须包含 stdio.h 一次即可。

应　该	不 应 该
当只须打印文本而不需要打印变量时，应使用 puts()函数，而不是 printf()函数。	不要在 puts()函数中使用转换说明符。

函数 puts()的语法如下：

```
#include <stdio.h>
puts( string );
```

函数 puts()将一个字符串复制到标准输出设备（通常是显示器屏幕）中。使用 puts()时，应包含标准输入/输出头文件（stdio.h）。

puts()函数在打印完字符串后自动换行。格式化字符串可以包含转义序列，表 7.1 列出了最常用的转义序列。

下面是两个使用 puts()函数的范例及其输出：

范例 1：

```
puts("This is printed with the puts() function!");
```

该范例的输出如下：

```
This is printed with the puts() function!
```

范例 2：

```
puts("This prints on the first line. \nThis prints on the second line.");
puts("This prints on the third line.");
puts("If these were printf()s, all four lines would be on two lines!");
```

该范例的输出如下：

```
This prints on the first line.
This prints on the second line.
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
This prints on the third line.
If these were printf()s, all four lines would be on two lines!
```

7.2　使用 scanf()函数输入数值数据

大多数程序需要将数据输出到屏幕上，也需要从键盘读取输入数据。从键盘读取数值数据的最灵活的方法是使用库函数 scanf()。

函数 scanf()按指定的格式从键盘读取数据，并将其赋给一个或多个变量。和 printf()一样，scanf()也使用一个格式化字符串描述输入的格式，该格式化字符串使用的转换说明符与 printf()函数相同。例如，下面的语句：

```
scanf("%d", &x);
```

从键盘读取一个十进制整数，并将其赋给整型变量 x。同样，下面的语句从键盘读取一个浮点数，并将其赋给变量 rate：

```
scanf("%f", &rate);
```

变量名之前的&有何用处呢？符号&是地址运算符，有关它的详细内容请参阅第 9 天的课程。就现在而言，您只须记住在 scanf()函数中，其参数列表中的每个数值变量名前必须包含&即可。

一条 scanf()语句可以输入多个值，您只须在格式化字符串中包含多个转换说明符并在参数列表中包含多个变量名（每个变量名之前必须有&）即可。下面的语句输入一个整数值和一个浮点数，并将它们分别赋给变量 x 和 rate：

```
scanf("%d %f", &x, &rate);
```

输入多个变量时，scanf()使用空白将输入隔开。空白可以是空格、制表符或换行符。格式化字符串中的每个转换说明符对应一个输入字段，并用空白标识每个字段的结束。

这给您提供了极大的灵活性。为响应前面的 scanf()函数，您可以输入：

```
10 12.45
```

也可以输入：

```
10          12.45
```

还可以这样输入：

```
10
12.45
```

只要值之间有空白，scanf()便能将每个值赋给相应的变量。



> 警告：使用 scanf()时应小心。如果您要读取一个字符，而用户输入一个数字；或者您要读取一个数字，而用户输入一个字符，结果将可能出乎用户的意料。

与今天讨论的其他函数一样，使用 scanf()函数的程序也必须包含头文件 stdio.h。程序清单 7.3 是一个使用 scanf()的范例，第 15 天的课程将对该函数做更详细的讨论。

程序清单 7.3　　scanit.c：使用 scanf()读取数值

```
1:  /* Demonstration of using scanf() */
2:
3:  #include <stdio.h>
4:
5:  #define QUIT 4
6:
7:  int get_menu_choice( void );
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
8:
9:  int main( void )
10: {
11:     int   choice    = 0;
12:     int   int_var   = 0;
13:     float float_var = 0.0;
14:     unsigned unsigned_var = 0;
15:
16:     while (choice != QUIT)
17:     {
18:        choice = get_menu_choice();
19:
20:        if (choice == 1)
21:        {
22:           puts("\nEnter a signed decimal integer (i.e. -123)");
23:           scanf("%d", &int_var);
24:        }
25:        if (choice == 2)
26:        {
27:           puts("\nEnter a decimal floating-point number\
28:                (e.g. 1.23)");
29:           scanf("%f", &float_var);
30:        }
31:        if (choice == 3)
32:        {
33:           puts("\nEnter an unsigned decimal integer \
34                (e.g. 123)" );
35:           scanf( "%u", &unsigned_var );
36:        }
37:     }
38:     printf("\nYour values are: int: %d  float: %f  unsigned: %u \n",
39:                              int_var, float_var, unsigned_var );
40:
41:    return 0;
42: }
43:
44: int get_menu_choice( void )
45: {
46:    int selection = 0;
47:
48:    do
49:    {
50:       puts( "\n1 - Get a signed decimal integer" );
51:       puts( "2 - Get a decimal floating-point number" );
52:       puts( "3 - Get an unsigned decimal integer" );
53:       puts( "4 - Quit" );
54:       puts( "\nEnter a selection:" );
55:
56:       scanf( "%d", &selection );
57:
58:    }while ( selection < 1 || selection > 4 );
59:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
60:     return selection;
61: }
```

该程序的运行情况如下：

```
1 - Get a signed decimal integer
2 - Get a decimal floating-point number
3 - Get an unsigned decimal integer
4 - Quit

Enter a selection:
1

Enter a signed decimal integer (e.g. -123)
-123

1 - Get a signed decimal integer
2 - Get a decimal floating-point number
3 - Get an unsigned decimal integer
4 - Quit

Enter a selection:
3

Enter an unsigned decimal integer (e.g. 123)
321

1 - Get a signed decimal integer
2 - Get a decimal floating-point number
3 - Get an unsigned decimal integer
4 - Quit

Enter a selection:
2

Enter a decimal floating point number (e.g. 1.23)
1231.123

1 - Get a signed decimal integer
2 - Get a decimal floating-point number
3 - Get an unsigned decimal integer
4 - Quit

Enter a selection:
4

Your values are: int: -123  float: 1231.123047 unsigned: 321
```

分析：程序清单 7.3 使用了与程序清单 7.1 相同的菜单概念，区别在于 get_menu_choice()（第 44～61 行）更小些，但有必要对其做些说明。首先，使用的是函数 puts()，而不是 printf()。由于不需要打印任何变量，因此没有必要使用 printf()。由于使用的是 puts()，因此第 51～53 行不需要转义字符换行符。第 58 行也做了修改，使得允许用户输入 1～4 之间的值，因为现在有 4 个菜单选项。第 56 行没变，但现在更清晰。scanf() 读取一个十进制值，并将其赋给变量 selection。在第 60 行，函数 get_menu_choice()将 selection 的值返回给调

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

用它的程序。

程序清单 7.1 和 7.3 使用的 main()函数的结构相同。使用一条 if 语句检测函数 get_menu_choice()返回的值 choice，并根据 choice 的值打印一条消息，提示用户输入一个数字，然后使用 scanf()来读取用户输入的值。请注意第 23、29 和 35 行之间的差别，它们用于读取不同类型的值。第 12～14 行声明了合适类型的变量。

用户选择退出时，程序将打印用户最后输入的三种类型的值。如果用户没有输入值，则打印 0，因为第 12～14 行已经将这三种类型的变量分别初始化为 0。最后需要注意的是第 20～36 行：这里使用的 if 语句的结构并不好，使用一个 if...else 结构更合适。第 14 天的课程将介绍一条新的控制语句 switch，该语句提供了一种更好的选择。

应　该	不 应 该
应结合使用 printf()（或 puts()）和 scanf()。使用打印函数显示提示消息，提示用户输入您使用 scanf()要读取的值。	使用 scanf()时，别忘了在变量前加上地址运算符&。

函数 scanf()的语法如下：

```
#include <stdio.h>
scanf( format-string[,arguments,...]);
```

函数 scanf()在格式化字符串中使用转换说明符把值放置到变量参数中。参数必须是变量的地址，而不是实际的变量本身。对于数值变量，可以通过在变量名前加上地址运算符（&）来传递其地址。使用 scanf()时，必须包含头文件 stdio.h。

scanf()读取标准输入流（通常是键盘）中的输入字段，并将每个字段放到一个参数中。放置信息时，scanf()将根据格式化字符串中相应的说明符指定的格式对信息进行转换。每个参数都必须有一个对应的转换说明符。表 7.2 列出了最常用的转换说明符。下面是两个使用 scanf()函数的例子：

范例 1：

```
int x, y, z;
scanf( "%d %d %d", &x, &y, &z);
```

范例 2：

```
#include <stdio.h>
int main( vcid )
{
    float y;
    int x;
    puts( "Enter a float, then an int" );
    scanf( "%f %d", &y, &x);
    printf( "\nYou entered %f and %d ", y, x );
    return 0;
}
```

7.3　三字符序列

介绍完使用 printf()和 scanf()等函数来读写信息的基本知识后，接下来介绍今天课程的最后一个主题。这个主题并非关于读写信息的，而是源代码中的特殊字符序列，这些序列有特殊的含义。这些特殊的序列被称为三字符序列。



注意：您可能永远都不会使用三字符序列。这里介绍它们旨在当您无意间在代码中使用了一个三字符序列，而它们按下面的方式被自动转换时，您不会感到莫名其妙。

三字符序列类似于前面介绍的转义序列。它们之间最大的差别在于，三字符序列在编译器编译源代码时

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

被解释。编译器对源代码中的所有三字符序列进行转换。

三字符序列以两个问号（??）打头，表 7.3 列出了 ANSI 标准规定的三字符序列。根据该标准，不应存在其他的三字符序列。

表 7.3　　三字符序列

编　码	对应的字符
??=	#
??(	[
??/	\
??)	]
??'	^
??<	{
??!	\|
??>	}
??-	~

源代码中的所有三字符序列将被转换为相应的字符，即使它位于字符串中。例如：

```
printf("??(WOW??)");
```

将被转换为：

```
printf("[WOW]");
```

如果包含更多的问号，则其余的问号不变。例如：

```
printf("???-");
```

将被转换为：

```
printf("?~");
```

7.4　总　结

阅读完今天的课程后，您便可以编写自己的程序了。通过结合使用函数 printf()、puts()、scanf()以及前几天介绍的程序控制语句，便可以编写出简单的程序。

在屏幕上显示信息是通过函数 printf()和 puts()完成的。puts()函数只能显示文本消息，而 printf()能够显示文本消息和变量。这两个函数都使用转义序列来显示特殊的字符以及控制打印。

scanf()函数从键盘读取一个或多个数值，并根据转换说明符对这些值进行解释。其中每个值都被赋给一个变量。

今天课程的最后介绍了三字符序列，它们是特殊的代码，将被转换为相应的字符。

7.5　问与答

问：既然 printf()的功能强于 puts()，为何要使用 puts()？

答：正是由于 printf()的功能更强，因此开销也更大。编写小型、高效的程序，或程序很大、资源变得非常宝贵时，您要充分利用 puts()的开销更小这一优点。通常，应使用最简单的可用资源。

问：使用函数 printf()、puts()或 scanf()时，为何需要包含头文件 stdio.h？

答：stdio.h 中有标准输入/输出函数的原型，而 printf()、puts()和 scanf()都属于这样的标准函数。如果使用这些函数的程序没有包含头文件 stdio.h，编译器将生成错误和警告。

问：在 scanf()函数中，如果不在变量名前加上地址运算符（&），将发生什么情况？

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

答：这是人们容易犯的错误之一。如果遗漏了地址运算符，结果将是不可预测的。阅读第 9 和 13 天有关指针的内容后，您会对此有更深入的理解。就现在而言，只需知道这一点即可，即如果遗漏了地址运算符，scanf()就不会把输入的信息存储到变量中，可将其存储到内存的其他地方。这可能导致任何结果，最明显的是锁定计算机，您必须重新启动机器。

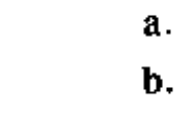
7.6 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

7.6.1 小测验

1. 函数 puts()和 printf()之间的区别是什么？
2. 使用 printf()时，必须包含哪个头文件？
3. 下述转义序列的含义分别是什么？
 a. \\
 b. \b
 c. \n
 d. \t
 e. \a
4. 要打印下述内容，应分别使用什么转换说明符？
 a. 字符串；
 b. 有符号的十进制整数；
 c. 十进制浮点数。
5. 在 puts()的字面文本中使用下述序列时，它们之间的区别是什么？
 a. b
 b. \b
 c. \
 d. \\

7.6.2 练习



注意：从今天的课程开始，将包含一些这样的练习，即要求您编写一个执行特定任务的程序。由于完成这样的工作有多种方式，因此附录 F 中提供的答案并非是唯一正确的。如果您能够编写执行所需任务的代码，则很好；如果有困难，请参考后面的答案，以获取帮助。答案中包含尽可能少的注释，这样让您能够有机会弄明白它们的工作原理。

1. 编写一条换行的 pirntf()语句和 puts()语句。
2. 编写一条 scanf()语句，该语句可用来读取一个字符、一个无符号十进制整数和另一个字符。
3. 编写读取并打印一个整数的语句。
4. 修改练习 3，使其只能读取偶数（2、4、6 等）。
5. 修改练习 4，使之在用户输入数字 99 或输入了 6 个偶数后，返回用户输入的值。将这些值存储在数组中（提示：需要编写一个循环）。
6. 将练习 5 转换为一个可执行的程序。添加一个函数，该函数在一行中打印数组中的值，并使用制表符将它们分开（只打印被存储在数组中的值）。
7. 排错：请找出下述代码片段中的错误。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
printf( "Jack said, "Peter Piper picked a peck of pickled peppers."");
```

8. 排错：请找出下述程序中的错误。

```
int get_1_or_2( void )
{
        int answer = 0;
        while (answer < 1 || answer > 2)
        {
            printf(Enter 1 for Yes, 2 for No);
            scanf( "%f", answer );
        }
        return answer;
}
```

9. 完善程序清单 7.1 中的 print_report()函数，使之能够打印表 7.1 中的其他内容。

10. 编写一个这样的程序：从键盘读取两个浮点数，并打印它们的乘积。

11. 编写一个这样的程序：从键盘读取 10 个整数，并显示它们的和。

12. 编写一个从键盘读取整数，并将它们存储在数组中的程序。当用户输入 0 或达到数组末尾时，输入将结束。然后找到并显示该数组中的最大和最小的值（注意：这个练习比较难，因为本书还未完整地介绍数组。如果您做起来有困难，请在阅读完第 8 天的课程后，再尝试完成它）。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第一周复习

至此，您完成了学习使用 C 语言进行编程的第一周课程，对于如何输入程序以及使用编辑器和编译器应该得心应手。下面的程序将前 7 天课程中介绍的许多内容组合在了一起。

这部分内容不同于前面您完成的 Type & Run 中的程序，其中包含了分析。该程序涉及的每个主题都在前一周的课程中介绍过了，第二周复习和第三周复习中的内容也是如此。



注意：左边的编号指出了代码涉及的主题是在哪一天介绍的，如果您对某些代码不太明白，可参阅相应的课程，获取更详细的信息。

程序清单 R1.1　　week1.c：第一周复习的程序清单

```
CH 02    1: /* Program Name: week1.c                                    */
         2: /* Purpose:    Program to enter the ages and incomes of up */
         3: /*             to 100 people.  The program prints a report */
         4: /*             based on the numbers entered.               */
         5: /*-------------------------------------------------------- */
         6: /*--------------------*/
         7: /* included files     */
         8: /*--------------------*/
CH 02    9: #include <stdic.h>
        10:
CH 02   11: /*--------------------*/
        12: /* defined constants  */
        13: /*--------------------*/
        14:
CH 03   15: #define MAX   100
        16: #define YES   1
        17: #define NO    0
        18:
CH 02   19: /*--------------------*/
        20: /* variables          */
        21: /*--------------------*/
        22:
CH 03   23: long  income[MAX]; /* to hold incomes    */
        24: int   month[MAX], day[MAX], year[MAX]; /* to hold birthdays */
        25: int   x, y, ctr;   /* For counters       */
        26: int   cont;  /* For program control */
        27: long  month_total, grand_total;  /* For totals     */
        28:
CH 02   29: /*--------------------*/
```

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
          30: '* function prototypes*/
          31: /*------ ------------*/
          32:
CH  05    33: int main(void);
          34: int display_instructions(void);
          35: void get_data(void);
          36: void display_report(void);
          37: int continue_function(void);
          38:
          39: /*---  ----- ---------*/
          40: /* start of program  */
          41: /*--- ----- ----------*/
          42:
CH  02    43: int main(void)
          44: {
CH  05    45:    cont = display_instructions();
          46:
CH  04    47:    if ( cont == YES )
          48:     {
CH  05    49:        get_data();
CH  05    50:       display_report();
          51:     }
CH  04    52:    else
CH  07    53:       printf("\nProgram Aborted by User!\n\n");
          54:
          55:   return 0;
          56: }
CH  02    57: /*--- ------------------------------------------------ -*
          58:  *  Function:  display_instructions()                    *
          59:  *  Purpose:    This function displays information on how to*
          60:  *           use this program and asks the user to enter 0 *
          61:  *           to quit, or 1 to continue.                   *
          62:  *  Returns:   NO  - if user enters 0                     *
          63:  *             YES - if user enters any number other than 0 *
          64:  *-------------------------------------------------------*/
CH  05    65: int display_instructions( void )
          66: {
CH  07    67:    printf("\n\n");
          68:    printf("\nThis program enables you to enter up to 99 people\'s");
          69:    printf("\nincomes and birthdays.  It then prints the incomes by");
          70:    printf("\nmonth along with the overall income and overall average.");
          71:    printf("\n');
          72:
CH  05    73:    cont = continue_function();
CH  05    74:     return( cont );
          75: }
CH  02    76: /*---------------------------------------------------- *
          77:  *  Function:  get_data() *
          78:  *  Purpose: This function gets the data from the user. It*
          79:  *           continues to get data until either 100 people are *
          80:  *           entered, or until the user enters 0 for the month.*
          81:  *  Returns: nothing *
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
          82:  *Notes: This allows 0/0/0 to be entered for birthdays in  *
          83:  *       case the user is unsure.  It also allows for 31    *
          84:  *       days in each month.                                *
          85:  *-----------------------------------------------------*/
          86:
CH  05    87:  void get_data(void)
          88:  {
CH  06    89:    for ( cont = YES, ctr = 0; ctr < MAX && cont == YES; ctr++ )
          90:    {
CH  07    91:      printf("\nEnter information for Person %d.", ctr+1 );
          92:      printf("\n\tEnter Birthday:");
          93:
CH  06    94:       do
          95:       {
CH  07    96:         printf("\n\tMonth (0 - 12): ");
CH  07    97:         scanf("%d", &month[ctr]);
CH  06    98:      }while (month[ctr] < 0 || month[ctr] > 12 );
          99:
CH  06    100:      do
          101:      {
CH  07    102:        printf("\n\tDay (0 - 31): ");
CH  07    103:         scanf("%d", &day[ctr]);
CH  06    104:     }while ( day[ctr] <  0 || day[ctr] > 31 );
          105:
CH  06    106:      do
          107:      {
CH  07    108:        printf("\n\tYear (0 - 2002): ");
CH  07    109:        scanf("%d", &year[ctr]);
CH  06    110:     }while ( year[ctr] < 0 || year[ctr] > 2002 );
          111:
CH  07    112:     printf("\nEnter Yearly Income (whole dollars): ");
CH  07    113:      scanf("%ld", &income[ctr]);
          114:
CH  05    115:      cont = continue_function();
          116:   }
CH  07    117:   /* ctr equals the number of people that were entered.*/
          118: }
CH  02    119: /*-----------------------------------------------------     *
          120:  *  Function: display_report()                              *
          121:  * Purpose: This function displays a report to the screen    *
          122:  *  Returns:  nothing                                       *
          123:  *  Notes:    More information could be printed.            *
          124:  *-----------------------------------------------------     */
          125:
CH  05    125: void display_report()
          127: {
CH  04    128:   grand_total = 0;
CH  07    129:   printf("\n\n\n");                  /* skip a few lines   */
          130:   printf("\n        SALARY SUMMARY");
          131:   printf("\n        ==============");
          132:
CH  06    133:  for( x = 0; x <= 12; x++ ) /* for each month, including 0*/
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
          134:     {
          135:       month_total = 0;
CH  04    136:       for( y = 0; y < ctr; y++ )
CH  06    137:       {
          138:          if( month[y] == x )
CH  04    139:              month_total += income[y];
CH  04    140:       }
          141:       printf("\nTotal for month %d is %ld", x, month_total);
CH  07    142:       grand_total += month_total;
CH  04    143:     }
          144:    printf("\n\nReport totals:");
CH  07    145:    printf("\nTotal Income is %ld", grand_total);
          146:    printf("\nAverage Income is %ld", grand_total/ctr );
          147:
          148:    printf("\n\n* * * End of Report * * *\n");
          149: }
CH  02    150: /*-------------------------------------------------------*
          151:  * Function: continue_function()                         *
          152:  * Purpose:  This function asks the user if they wish to continue.  *
          153:  * Returns:  YES - if user wishes to continue             *
          154:  *           NO - if user wishes to quit                 *
          155:  *-------------------------------------------------------*/
          156:
CH  05    157: int continue_function( void )
          158: {
CH  07    159:    printf("\n\nDo you wish to continue? (0=NO/1=YES): ");
          160:    scanf( "%d", &x );
          161:
CH  06    162:    while( x < 0 || x > 1 )
          163:     {
CH  07    164:       printf("\n%d is invalid!", x);
          165:       printf("\nPlease enter 0 to Quit or 1 to Continue: ");
          166:       scanf("%d", &x);
          167:     }
CH  04    168:    if(x == 0)
CH  05    169:       return(NO);
CH  04    170:    else
CH  05    171:       return(YES);
          172: }
```

分析：完成第一天课程和第二天课程中的小测验和练习后，您应该能够输入并编译该程序。该程序中的注释比本书中所有的程序清单都多，实际的C语言程序通常都包含这样的程序。您尤其应该注意程序开头以及每个主要函数之前的注释。第1～5行的注释对整个程序做了概要地说明，其中包括程序的名称。有些程序员还会提供诸如程序的作者、使用的编译器、程序的版本号、被链接到程序中的库以及程序的创建日期等信息。每个函数前面的注释描述了函数的用途、可能的返回值、函数的调用方式以及其他与该函数相关的信息。

第1～5行的注释指出，在该程序中，最多可以输入100个人的信息。程序调用display_instructions()（第45行），使用户能够输入数据。该函数显示有关如何使用该程序的说明，并询问用户想继续还是退出。从第67～71行可知，该函数使用了第7天课程介绍的printf()函数打印这些说明。

在第157～172行，continue_function()使用了本周最后一天介绍的一些特性。该函数询问用户是否要继续（第159行）。该函数还使用了第6天课程介绍的while语句来验证用户输入的答案是否是0或1，如果不

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

是函数将不断提示，让用户做出响应。收到合适的答案后，if...else 语句将返回常量 YES 或 NO。

该程序的核心为函数 get_data()和 display_report()。函数 get_data()提示用户输入数据，并将它们存储到程序开头附近声明的数组中。第 89 行使用一条 for 语句来提示用户输入数据，直到 cont 不等于常量 YES（从 continue_function()返回的值）或者计数器 ctr 大于或等于数组可包含的最大元素数目（MAX）。程序检查用户输入的所有信息，确保它是合适的。例如，第 94～98 行提示用户输入月份。程序只接受 0～12 的数字，如果用户输入的数字大于 12，程序将再次提示用户输入月份。第 115 行调用 continue_function()检查用户是否要继续添加数据。

当用户用 0 来响应 continue_function()函数或输入的信息组数达到最大（MAX）时，程序将返回到 main()函数中的第 50 行，该行调用 display_report()函数。display_function()函数位于第 119～149 行，它将一个报告显示到屏幕上。该报告使用一个嵌套的 for 循环来计算每个月的总收入以及所有月份的总收入之和。您可能觉得该报告很复杂，如果是这样，请复习第 6 天课程中有关嵌套语句的内容。作为程序员，您创建的很多报告都要比它复杂。

该程序使用了您在本周学到的知识。对于一周时间而言，这些内容算很多了，但您学完了。使用本周所学的知识，您将能够使用 C 语言编写出自己的程序，但能够完成的工作有限。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第二周课程

完成第一周学习使用 C 语言编写程序的课程后，您应该能够熟练地输入程序以及使用编辑器和编译器。

本周课程的内容

本周的内容很多，您将学习 C 语言中的很多核心特性，包括如何使用数值数组和字符数组、将字符变量类型扩展到数组和字符串以及使用结构将不同的变量类型组合在一起。

本周的课程将以第一周介绍的内容为基础，介绍其他的程序控制语句，详细地解释函数。

第 9 天和第 12 天的课程重点介绍对于使用 C 语言而言至关重要的概念，您应额外花些时间来使用指针及其基本功能。

除了作用域和指针等重要主题外，第 8 天和第 11 天的课程介绍如何使用数组、结构和共用体等来存储信息。第 10 天的课程介绍如何使用字符和字符串；而第 14 天的课程介绍如何控制信息的打印和显示。

阅读完第一周的课程后，您学会了如何编写简单的程序；而阅读完本周的课程后，您将能够编写复杂的程序，这些程序几乎能够完成任何任务。

www.plcworld.cn

第 8 天课程　使用数值数组

数组是C程序经常使用的一种数据存储方式。第6天的课程对数组做了简要的介绍，今天您将学习以下内容：

- 数组是什么？
- 一维数值数组和多维数值数组的定义。
- 如何声明和初始化数组。

8.1　数组是什么

数组是一组数据存储位置，其中每个位置的名称相同，存储的数据类型也相同。数组中的数据存储位置被称为数组元素。程序为何需要使用数组呢？这个问题可以通过一个范例来回答。

如果您要记录2003年的营业支出并整理每个月的收据，可以把每个月的收据放在一个单独的文件夹中，但使用一个带12个间格的文件夹会更方便。

将这个范例扩展到计算机编程。假设您要设计一个记录营业开支的程序，程序可以声明12个不同的变量，每个变量记录一个月的总支出。这类似于用12个不同的文件夹收藏收据。然而，一种优秀的编程习惯是，使用一个包含12个元素的数组，将每个月的总支出存储在相应的数组元素中。这种方法类似于用一个包含12个间格的文件夹来收藏收据。图8.1说明了使用不同的变量和数组之间的区别。

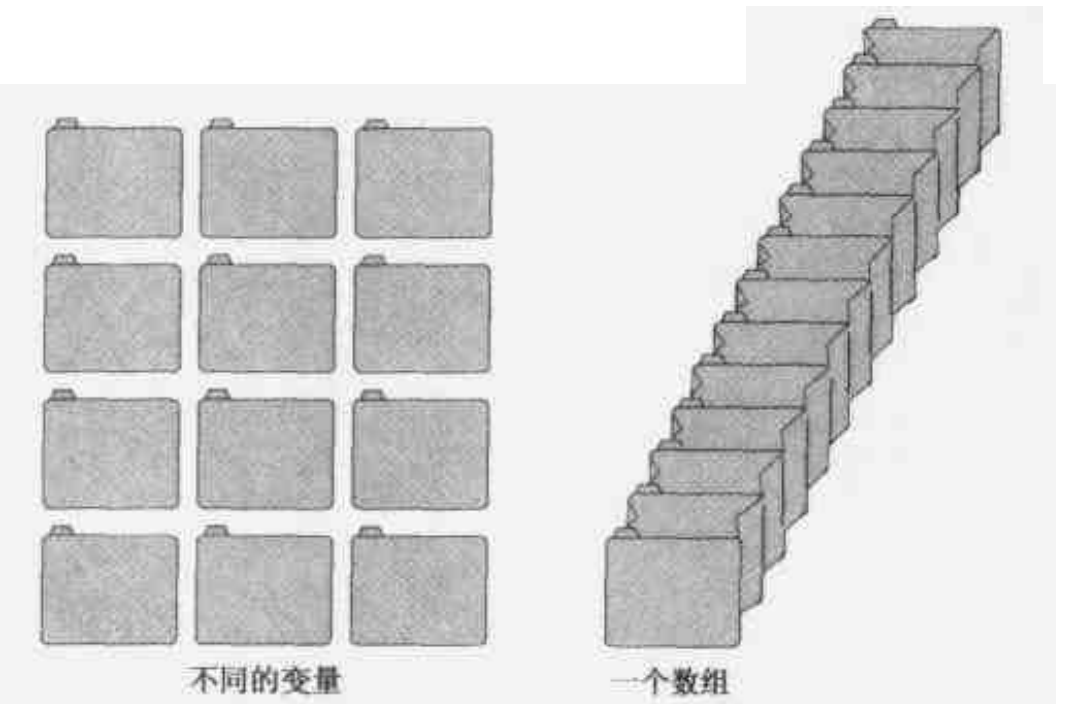


图8.1　变量就像不同的文件夹，而数组就像一个有很多间格的文件夹

8.1.1　一维数组

一维数组只有一个下标。下标是数组名后面位于方括号中的数字，可用于指出数组中各个元素的编号。下面用一个范例来阐明这一点。对于营业开支程序而言，您可以使用下面的代码行声明一个float数组：

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
float expenses[12];
```

该数组名为 expenses，包含 12 个元素，其中每个元素都相当于一个 float 变量。

数组可以是任何数据类型。C 语言中的数组元素总是从 0 开始编号，因此 expenses 的 12 个元素的编号为 0~11。在前面的范例中，一月份的总支出将存储在 expenses[0]中，二月份的存储在 expenses[1]中，依此类推。十二月份的总支出将存储在 expenses[11]中。

当您声明数组后，编译器将留出一块足够容纳整个数组的内存。各个数组元素在内存中被顺序存储，如图 8.2 所示。

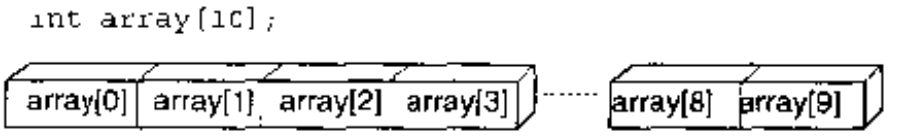


图 8.2　数组元素在内存单元中被顺序存储

在源代码的什么位置声明数组至关重要。和不属于数组的变量一样，声明所在的位置也会影响程序如何使用数组。有关声明位置的影响将在第 12 天的课程中详细介绍。现在，只需将数组声明和其他变量声明放在一起即可。

数组元素可用于能够使用同类型变量的任何位置。数组的各个元素是通过后面带下标（用方括号括起）的数组名访问的。例如，下面的语句将 89.95 存储在数组的第二个元素中（别忘了，第一个元素是 expenses[0]，而不是 expenses[1]）：

```
expenses[1] = 89.95;
```

同样，下面的语句：

```
expenses[10] = expenses[11];
```

将数组元素 expenses[11]的值复制给数组元素 expenses[10]。引用数组元素时，数组下标可以是字面常量，就像上面的范例那样。然而，程序经常使用整型变量、表达式，甚至另一个数组元素作为下标。下面是一些这样的例子：

```
float expenses[100];
int a[10];
/* additional statements go here */
expenses[i] = 100;          // i is an integer variable
expenses[2 + 3] = 100;      // equivalent to expenses[5]
expenses[a[2]] = 100;       // a[] is an integer array
```

有必要对最后一个范例做些解释。例如，如果有一个名为 a[]的数组，且 a[2]的值为 8，则代码：

```
expenses[a[2]]
```

与下面的代码等效：

```
expenses[8];
```

使用数组时，请切记元素编号方案：在一个包含 n 个元素的数组中，允许的下标为 0 到 n-1。如果使用下标 n，则可能发生错误。C 编译器不对程序使用的下标是否超界进行判断，程序将顺利地编译和链接，但下标超界通常会导致错误的结果。



警告：请记住，数组元素从 0（而不是 1）开始编号，另外，最后一个元素的编号比数组包含的元素数小 1。例如，在包含 10 个元素的数组中，元素的编号为 0~9。

有时候，您可能希望包含 n 个元素的数组中的元素编号好像是 1-n。例如，在前面的范例中，一种更自然的方法是将一月份的总支出存储在 expense[1]中，二月份的总支出存储在 expense[2]中，依此类推。为此，最简单的方法是，声明一个元素数目比所需的量多 1 的数组，且不使用编号为 0 的元素。就这个例子而言，您可以像下面那样声明一个数组。您可以将一些相关的数组存储在 0 号元素中（可能是全年的总支出）。

```
float expenses[13];
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 8.1 中的程序 expenses.c 演示了如何使用数组。这个程序很简单，没有什么实用价值，但它演示了如何使用数组。

程序清单 8.1　　expenses.c：使用数组

```
1:  /* expenses.c - Demonstrates use of an array */
2:
3:  #include <stdio.h>
4:
5:  /* Declare an array to hold expenses, and a counter variable */
6:
7:  float expenses[13];
8:  int count;
9:
10: int main( void )
11: {
12:     /* Input data from keyboard into array */
13:
14:     for (count = 1; count < 13; count++)
15:     {
16:        printf("Enter expenses for month %d: ", count);
17:        scanf("%f", &expenses[count]);
18:     }
19:
20:     /* Print array contents */
21:
22:     for (count = 1; count < 13; count++)
23:     {
24:        printf("Month %d = $%.2f\n", count, expenses[count]);
25:     }
26:     return 0;
27: }
```

该程序的运行情况如下：

```
Enter expenses for month 1: 100
Enter expenses for month 2: 200.12
Enter expenses for month 3: 150.50
Enter expenses for month 4: 300
Enter expenses for month 5: 100.50
Enter expenses for month 6: 34.25
Enter expenses for month 7: 45.75
Enter expenses for month 8: 195.00
Enter expenses for month 9: 123.45
Enter expenses for month 10: 111.11
Enter expenses for month 11: 222.20
Enter expenses for month 12: 120.00
Month 1 = $100.00
Month 2 = $200.12
Month 3 = $150.50
Month 4 = $300.00
Month 5 = $100.50
Month 6 = $34.25
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
Month 7 = $45.75
Month 8 = $195.00
Month 9 = $123.45
Month 10 = $111.11
Month 11 = $222.20
Month 12 = $120.00
```

分析：运行该程序时，它将提示您输入 1～12 月的支出。输入的值将存储在数组中。您必须输入每个月的值，输入 12 个值后，数组的内容将被显示在屏幕上。

该程序的流程与以前的程序类似。第 1 行是注释，描述了程序的功能。注意，注释中还包含程序的名称，这样您知道正在查看的是哪个程序。当您核对程序清单的打印输出时，这很有用。

第 5 行是另一个注释，它解释了被声明的变量。第 7 行声明了一个包含 13 个元素的数组。对这个程序而言，只需要 12 个元素，每个月一个，但却声明了 13 个。第 14～18 行的 for 循环忽略了 0 号元素。这样程序将使用 1～12 号元素，这些编号与 12 个月直接相关。第 8 行声明了一个名为 count 的变量，在整个程序中，该变量被用作计数器和数组索引。

从第 10 行开始，是程序的 main()函数。正如前面指出的，该程序使用一个 for 循环来打印一条消息，并接受 12 个月中每个月的值。第 17 行的 scanf()函数使用了一个数组元素。在第 7 行，数组的类型被声明为 float，因此这里使用转换说明符%f。另外，在数组元素的前面也加上了地址运算符&，就像它是一个常规的 float 变量，而不是数组元素。

第 22～25 行是第二个 for 循环，它打印刚才输入的值。printf()函数中还包含另一个格式化命令，以便整齐有序地打印支出。就现在而言，只需知道%.2f 打印包含两位小数的浮点数即可。更多的格式化命令将在第 14 天的课程中做更为详细的介绍。

应　该	不 应 该
存储多个同类型的值时，应声明一个数组，而不是声明多个变量。例如，如果要存储一年中各个月的总销售量，应声明一个包含 12 个元素的数组，而不是创建 12 个变量。	别忘了，数组的下标是从 0 开始的。

8.1.2　多维数组

多维数组有多个下标。二维数组有两个下标，三维数组有三个下标，依此类推。在 C 语言中，对数组的维数没有限制（但对数组的总长度有限制，请参阅今天课程后面的内容）。

例如，编写一个下国际象棋的程序。棋盘有 64 格，分 8 行 8 列排列。程序可以使用一个二维数组来表示棋盘，如下所示：

```
int checker[8][8];
```

该数组包含 64 个元素：checker[0][0]、checker[0][1]、checker[0][2]...checker[7][6]、checker[7][7]。该二维数组的结构如图 8.3 所示。

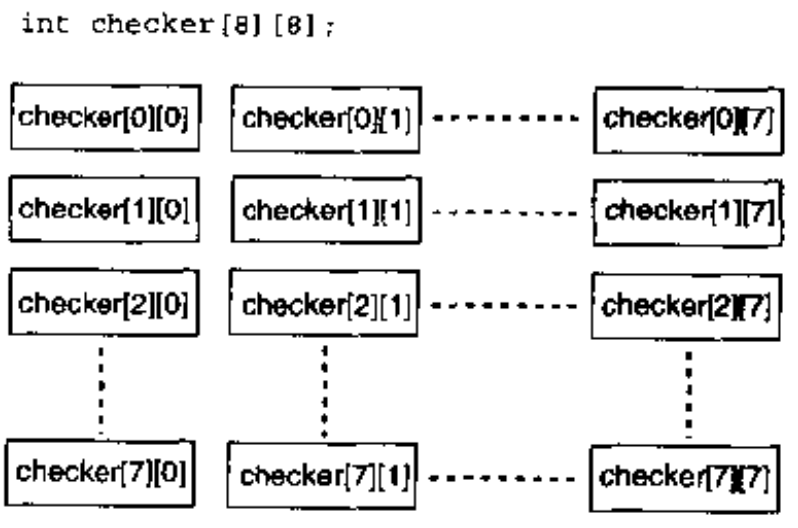


图 8.3　二维数组由行和列组成

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

同样，可将三维数组看作一个立方体。至于四维（或维数更多的）数组，就只能靠您发挥想象了。所有数组（不管它有多少维）在内存中都是顺序存储的，有关数组存储的更详细的信息，请参阅第 15 天的课程。

8.2 命名和声明数组

给数组命名的规则与变量相同，这些规则在第 3 天的课程中介绍过了。数组名必须是唯一的，不能用作其他数组或标识符（变量、常量等等）的名称。您可能猜到了，数组的声明规则和非数组变量相同，只是数组名后必须带元素数目，并用方括号括起。

声明数组时，可以使用字面常量来指定元素数目（就像前面的范例那样），也可以使用#define 编译指令定义的符号常量来指定。因此，代码：

```
#define MONTHS 12
int array[MONTHS];
```

与下面的语句等价：

```
int array[12];
```

然而，对于大多数编译器来说，不能使用 const 关键字定义的符号常量来指定元素数目：

```
const int MONTHS = 12;
int array[MONTHS];          /* Wrong! */
```

程序清单 8.2 中的程序 grades.c 再次演示了如何使用数组，该程序使用一个数组来存储 10 个分数。

程序清单 8.2　　grades.c：在一个数组中存储 10 个分数

```
1:  /*grades.c - Sample program with array */
2:  /* Get 10 grades and then average them */
3:
4:  #include <stdio.h>
5:
6:  #define MAX_GRADE 100
7:  #define STUDENTS  10
8:
9:  int grades[STUDENTS];
10:
11: int idx;
12: int total = 0;          /* used for average */
13:
14: int main( void )
15: {
16:     for( idx=0;idx< STUDENTS;idx++)
17:     {
18:         printf( "Enter Person %d'sgrade: ",idx +1)
19:         scanf( "%d", &grades[idx] );
20:
21:         while ( grades[idx] > MAX_GRADE )
22:         {
23:             printf( "\nThe highest grade possible is %d",
24                      MAX_GRADE );
25:             printf( "\nEnter correct grade: " );
26:             scanf( "%d", &grades[idx] );
27:         }
28:
29:         total += grades[idx];
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
30:     }
31:
32:     printf( "\n\nThe average score is %d\n", ( total / STUDENTS) );
33:
34:     return (0);
35: }
```

该程序的运行情况如下：

```
Enter Person 1's grade:  95
Enter Person 2's grade:  100
Enter Person 3's grade:  60
Enter Person 4's grade:  105

The highest grade possible is 100
Enter correct grade: 100
Enter Person 5's grade:  25
Enter Person 6's grade:  0
Enter Person 7's grade:  85
Enter Person 8's grade:  85
Enter Person 9's grade:  95
Enter Person 10's grade:  85

The average score is 73
```

分析：和 expenses.c 一样，该程序也提示用户输入值。它提示用户输入 10 个人的分数，然后打印平均分数，而不是每个分数。

正如前面指出的，数组的命名方式和常规变量相同。第 19 行将该程序使用的数组命名为 grades，从该数组的名称可知，它用于存储分数。第 6 和 7 行定义了两个常量：MAX_GRADE 和 STUDENTS。可以很容易地修改这些常量的值。由于 STUDENTS 为 10，因此数组 grades 有 10 个元素。第 11 和 12 行声明了其他两个变量：idx 和 total。idx 是 index 的简称，被用作计数器和下标。所有分数的总和被存储在 total 中。

该程序的核心是第 16～30 行的 for 循环。for 语句将 idx 的值初始化为 0——数组的第一个下标。然后，不断循环，直到 idx 大于或等于学生数目。每循环一次，idx 的值便加 1。每次循环时，程序提示用户输入学生的分数（第 18 和 19 行）。在第 18 行，将 idx 加 1，以便从 1 数到 10，而不是从 0 数到 9。由于数组的第一个下标为 0，因此第一个分数存储在 grade[0]中。程序提示用户输入第一个（而不是第 0 个）学生的分数，以防使用户迷惑。

第 21～27 行是个 while 循环，它被嵌套在 for 循环中。它确保用户输入的分数不大于最大值（MAX_GRADE）。如果用户输入的分数过高，程序将提示用户输入正确的分数。您应尽可能地检查输入的数据。

第 29 行将输入的分数加入到 total 计数器中。第 32 行使用总分数来打印平均分数（total/STUDENTS）。

应　该	不应该
应使用#define 语句来定义常量，这样便可以使用它来声明数组。然后，您可以轻松地修改数组包含的元素数目。例如，在 grades.c 中，可以在#define 语句中修改 STUDENTS 的值，而无需对程序做其他任何修改。	应尽可能不要使用超过三维的数组。多维数组很容易变得非常大。

8.2.1　初始化数组

声明数组时可以初始化数组的部分元素或所有元素，只需在数组声明的后面加上等号和一系列的值，用花括号将这些值括起，并使用逗号将它们分开即可。列表中的值将依次被赋给数组中的元素（从 0 号元素开

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

始）。

请看下面的代码：

```
int array[4] = { 100, 200, 300, 400 };
```

在这个例子中，100、200、300 和 400 分别被赋给数组元素 array[0]、array[1]、array[2]和 array[3]。

如果省略数组大小，编译器将创建一个刚好能够存储初始化值的数组。因此，下面的语句与上述数组声明语句等价：

```
int array[] = { 100, 200, 300, 400 };
```

然而，初始化值的数目可以少于元素个数，如下面的范例所示：

```
int array[10] = { 1, 2, 3 };
```

如果不显式地初始化数组元素，则无法确定程序运行时该元素包含的值。如果初始化值的数目多于元素数目，编译器将产生错误。根据 ANSI 标准，没有被初始化的元素将被设置为 0。



提示：不要依赖于编译器来初始化元素的值。最好显式地初始化元素，以便您知道初始化值。

8.2.2 初始化多维数组

多维数组也可被初始化。初始化中的值将被依次赋给数组元素（最后一个下标首先变化）。例如：

```
int array[4][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

将导致如下结果：

```
array[0][0] is equal to 1
array[0][1] is equal to 2
array[0][2] is equal to 3
array[1][0] is equal to 4
array[1][1] is equal to 5
array[1][2] is equal to 6
array[2][0] is equal to 7
array[2][1] is equal to 8
array[2][2] is equal to 9
array[3][0] is equal to 10
array[3][1] is equal to 11
array[3][2] is equal to 12
```

初始化多维数组时，可以添加额外的花括号，将初始化值分组，并将它们放在多行中，这样源代码将更为清晰。下面的初始化语句与前面的等价：

```
int array[4][3] = { { 1, 2, 3 } , { 4, 5, 6 } ,
{ 7, 8, 9 } , { 10, 11, 12 } };
```

必须使用逗号将初始化值分开，即使它们之间有花括号。另外，花括号必须成对出现——每个左括号都有对应的右括号，否则编译器将报错。

下面来看一个例子，它演示了数组的优点。程序清单 8.3 中的 random.c 程序创建了一个包含 1000 个元素的三维数组，并使用随机数来填充它，然后将数组元素显示在屏幕上。请想象一下，如果用非数组变量来完成这样的任务，需要编写多少行的源代码。

该程序使用了一个新的库函数——getchar()，它从键盘读取一个字符。在程序清单 8.3 中，getchar()让程序暂停，直到用户按下 Enter 键。有关该函数的细节，请参阅第 14 天的课程。

程序清单 8.3　　random.c：创建一个多维数组

```
1:   /* random.c - Demonstrates using a multidimensional array */
2:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:  /* Declare a three-dimensional array with 1000 elements */
6:
7:  int random_array[10][10][10];
8:  int a, b, c;
9:
10: int main( void )
11: {
12:     /* Fill the array with random numbers. The C library */
13:     /* function rand() returns a random number. Use one */
14:     /* for loop for each array subscript. */
15:
16:     for (a = 0; a < 10; a++)
17:     {
18:         for (b = 0; b < 10; b++)
19:         {
20:             for (c = 0; c < 10; c++)
21:             {
22:                 random_array[a][b][c] = rand();
23:             }
24:         }
25:     }
26:
27:     /* Now display the array elements 10 at a time */
28:
29:     for (a = 0; a < 10; a++)
30:     {
31:         for (b = 0; b < 10; b++)
32:         {
33:             for (c = 0; c < 10; c++)
34:             {
35:                 printf("\nrandom_array[%d][%d][%d] = ", a, b, c);
36:                 printf("%d", random_array[a][b][c]);
37:             }
38:             printf("\nPress Enter to continue, CTRL-C to quit.");
39:
40:             getchar();
41:         }
42:     }
43:     return 0;
44: }   /* end of main() */
```

该程序的输出如下所示：

```
random_array[0][0][0] = 346
random_array[0][0][1] = 130
random_array[0][0][2] = 10982
random_array[0][0][3] = 1090
random_array[0][0][4] = 11656
random_array[0][0][5] = 7117
random_array[0][0][6] = 17595
random_array[0][0][7] = 6415
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
random_array[0][0][8] = 22948
random_array[0][0][9] = 31126
Press Enter to continue, CTRL-C to quit.

random_array[0][1][0] = 9004
random_array[0][1][1] = 14558
random_array[0][1][2] = 3571
random_array[0][1][3] = 22879
random_array[0][1][4] = 18492
random_array[0][1][5] = 1360
random_array[0][1][6] = 5412
random_array[0][1][7] = 26721
random_array[0][1][8] = 22463
random_array[0][1][9] = 25047
Press Enter to continue, CTRL-C to quit
...     ...
random_array[9][8][0] = 6287
random_array[9][8][1] = 26957
random_array[9][8][2] = 1530
random_array[9][8][3] = 14171
random_array[9][8][4] = 6951
random_array[9][8][5] = 213
random_array[9][8][6] = 14003
random_array[9][8][7] = 29736
random_array[9][8][8] = 15028
random_array[9][8][9] = 18966
Press Enter to continue, CTRL-C to quit.

random_array[9][9][0] = 28559
random_array[9][9][1] = 5268
random_array[9][9][2] = 20182
random_array[9][9][3] = 3633
random_array[9][9][4] = 24779
random_array[9][9][5] = 3024
random_array[9][9][6] = 10853
random_array[9][9][7] = 28205
random_array[9][9][8] = 8930
random_array[9][9][9] = 2873
Press Enter to continue, CTRL-C to quit.
```

分析：第 6 天的课程中有一个使用嵌套 for 语句的程序，而上述程序包含两个嵌套 for 循环。详细介绍 for 语句之前，请看第 7 和 8 行，它们声明了 4 个变量。第一个为数组 random_array，用于存储随机数。random_array 是一个三维的 int 数组，每维的长度皆为 10，因此有 1000 个类型为 int 的元素。如果不使用数组，则必须使用 1000 个名称唯一的变量！第 8 行声明了三个变量：a、b、c，用于控制 for 循环。

该程序的第 4 行还包含了（标准库的）头文件 stdlib.h，提供第 22 行使用的函数 rand()的原型。

程序的主体是两个嵌套的 for 语句。第一个位于第 16～25 行，第二个位于第 29～42 行。这两个嵌套 for 循环的结构相同，其工作方式与程序清单 6.2 中的循环类似，但多了一层嵌套。在第一个嵌套 for 循环中，被重复执行的语句是第 22 行。第 22 行将函数 rand()返回的值赋给数组 random_array 的一个元素，其中 rand()是一个库函数，它返回一个随机数。

第 20 行将变量 c 的值从 0 递增到 9，从而遍历数组 random_array 最右边的下标。第 18 行遍历数组第二

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

个下标（b）的值。b 的值每修改一次，c 的值都从 0 递增到 9。第 16 行递增变量 a 的值，从而遍历数组第一个下标。该下标每变化一次，b 的值都从 0 递增到 9，而 b 的值每变化一次，c 的值都从 0 递增到 9。这样，该循环便将数组中的每个元素初始化为一个随机数。

第 29 ~42 行是第二个嵌套 for 循环，其工作原理与前一个嵌套 for 循环相同，但执行的任务是打印每个数组元素的值。每显示 10 个元素后，第 38 行便打印一条消息，等待用户按 Enter 键后继续显示。第 40 行使用 getchar()来处理用户按键，如果用户按下 Enter 键，程序将继续显示后面的元素。请运行该程序，并查看显示的值。

8.2.3　数组的最大长度

由于内存模型的工作方式，现在您不应创建超过 64KB 的数据变量。解释这种限制超越了本书的范围，但您不用担心：本书中的所有程序都没有超过这一限制。要了解更多的信息或避开这种限制，请参阅编译器的用户手册。通常，64KB 的数据空间足够了，尤其是对于您在阅读本书时将编写的相对简单的程序而言。如果您不使用其他变量，则一个数组可以占用全部的 64KB 数据存储空间；如果需要使用其他的变量，则必须根据需要分配可用的数据空间。



> 注意：有些操作系统不存在 64KB 的限制。例如 DOS 有这样的限制，而 Windows 没有。

数组的长度（单位为字节）取决于它包含的元素数目和每个元素的长度。元素长度取决于数组的数据类型和您使用的计算机。表 3.2 列出了每种数值数据类型的长度，出于方便的考虑，表 8.1 再次列出了这些内容。对于很多 PC 而言，这些数据类型长度都是正确的。

表 8.1　　在很多 PC 中，数值数据类型所需的存储空间

元素数据类型	元素长度（字节）
int	4
short	2
long	4
long long	8
float	4
double	8

要计算数组所需的存储空间，可将数组包含的元素数目乘以元素长度。例如，包含 500 个元素的 float 数组需要的存储空间为 500×4 = 2000 字节。

正如本书前面介绍的，在程序中可以使用运算符 sizeof 来确定变量所需的存储空间，sizeof 是一个单目运算符，而不是函数。它将变量名或数据类型名作为参数，并返回参数的长度（单位为字节）。程序清单 8.4 演示了如何使用 sizeof。

程序清单 8.4　　arraysize.c：使用运算符 sizeof 来确定数组所需的存储空间

```
1:  /* Demcnstrates the sizeof() operator */
2:
3:  #include <stdio.h>
4:
5:  /* Declare several 100-element arrays */
6:
7:  int intarray[100];
8:  float floatarray[100];
9:  double doublearray[100];
10:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
11: int main()
12: {
13:     /* Display the sizes of numeric data types */
14:
15:     printf("\n\nSize of int = %d bytes", sizeof(int));
16:     printf("\nSize of short = %d bytes", sizeof(short));
17:     printf("\nSize of long = %d bytes", sizeof(long));
18:     printf("\nSize of long long = %d bytes", sizeof(long long));
19:     printf("\nSize of float = %d bytes", sizeof(float));
20:     printf("\nSize of double = %d bytes", sizeof(double));
21:
22:     /* Display the sizes of the three arrays */
23:
24:     printf("\nSize of intarray = %d bytes", sizeof(intarray));
25:     printf("\nSize of floatarray = %d bytes",
26:           sizeof(floatarray));
27:     printf("\nSize of doublearray = %d bytes\n",
28:           sizeof(doublearray));
29:
30:     return 0;
31: }
```

在32位的Windows 3.1机器上运行上述程序时，输出如下：

```
Size of int = 2 bytes
Size of short = 2 bytes
Size of long = 4 bytes
Size of long long = 8 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of intarray = 200 bytes
Size of floatarray = 400 bytes
Size of doublearray = 800 bytes
```

在32位的Windows NT以及32位的Linux或UNIX机器上运行该程序时，输出将如下：

```
Size of int = 4 bytes
Size of short = 2 bytes
Size of long = 4 bytes
Size of long long = 8 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of intarray = 400 bytes
Size of floatarray = 400 bytes
Size of doublearray = 800 bytes
```

分析：请按第1天介绍的步骤输入并编译上述程序清单中的程序。该程序运行时，将显示3个数组和6个数值数据类型的长度（单位为字节）。

在第3天的课程中，您运行了一个简单的程序；然而上述程序使用sizeof来确定数组的长度。第7～9行声明了三个类型各不相同的数组。第23～27行打印每个数组的长度。数组的长度等于其数据类型的长度乘以元素数目。例如，如果数据类型int的长度为4字节，则数组intarray的长度为400（4×100）字节。请运行该程序，并查看其显示的值。从上述输出可知，在不同的机器（操作系统）中，数据类型的长度可能不同。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com



> 提示：可以将数组长度除以元素长度来确定数组包含的元素数目。例如，对于程序清单 8.4 中的数组 doublearray，可以使用下面的代码计算其包含的元素数目：
>
> ```
> ArraySize = sizeof(doublearray) / sizeof(double);
> ```

8.3 总　结

今天的课程介绍了数值数组——一种功能强大的数据存储方式，让您能够将类型相同的数据项组合在一起，并使用相同的名称。数组中的数据项（元素）用数组名后面的下标标识。涉及到重复处理数据的计算机编程任务适合使用数组来完成。

和非数组变量一样，使用数组之前也必须声明；另外，还可以在声明数组时对数组元素进行初始化。

8.4 问与答

问：使用数组时，如果下标大于数组包含的元素数，将发生什么情况？

答：如果使用的下标超出了数组声明的界限，程序将能够通过编译，甚至能够运行。然而，结果可能是不可预测的。当这种错误引发问题时，查找起来很困难，因此初始化和访问数组元素时，一定要小心。

问：使用未被初始化的数组时，将发生什么情况？

答：这不会导致编译错误。如果数组未被初始化，则其元素的值将是不确定的，因此结果可能出乎您的意料。请务必初始化变量和数组，这样您便能够知道它们的值。第 12 天的课程将介绍一个无需初始化的情况。就现在而言，还是谨慎行事为好。

问：数组可以有多少维？

答：正如今天的课程中指出的，数组可以有任意维数。维数越多，数组使用的数据存储空间越大。声明数组时，应以够用为原则，以免浪费存储空间。

问：有没有一次性初始化整个数组的简易方法？

答：必须初始化数组的每个元素。对于 C 语言初学者而言，最安全的初始化数组的方式是，在声明时进行初始化或使用 for 语句进行初始化。还有其他一些初始化数组的方式，但这超出了本书的范围。

问：可以将两个数组相加（相乘、相除或相减）吗？

答：如果您声明了两个数组，不能将它们相加，而必须分别对每个数组进行相加，练习 10 说明了这一点。然而，您可以创建一个将两个数组相加的函数，这种函数仍然需要分别将各个数组相加。

问：为何说使用数组优于分别使用变量？

答：使用数组，可以给一组类似的值提供一个名称。在程序清单 8.3 中，存储了 1000 个值。如果创建 1000 个变量，并将每个变量初始化为一个随机数，则需要输入大量的代码。使用数组时，这样的任务将非常简单。

问：编写程序时，如果不知道数组应多大，该如何办？

答：C 语言中有这样的函数，即让您能够在程序运行时动态地为变量和数组分配内存。这些函数将在第 15 天的课程中介绍。

8.5 作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

8.5.1 小测验

1. 数组可以是哪些数据类型？

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

2. 对于被声明为包含 10 个元素的数组，其第一个元素的下标是多少？
3. 对于被声明为包含 n 个元素的一维数组，其最后一个元素的下标是多少？
4. 如果程序试图使用超出界限的下标来访问数组元素，情况将如何？
5. 如何声明多维数组？
6. 下述语句声明的数组总共包含多少个元素：

```
int array[2][3][5][8];
```

7. 上述数组的第 10 元素的名称是什么？
8. 对于类型为 long 的数组 xyz，如何确定其包含的元素数目？

8.5.2 练习

1. 编写一行代码，它声明三个一维的 int 数组，这些数组的名称分别为 one、two 和 three，它们都包含 1000 个元素。
2. 编写一条语句，它声明一个包含 10 个元素的 int 数组，并将所有的元素初始化为 1。
3. 编写将下述数组的所有元素初始化为 88 的代码：

```
int eightyeigth[88];
```

4. 编写将下述数组的所有元素初始化为 0 的代码：

```
int stuff[12][10];
```

5. 排错：下述代码片段有何错误？

```
int x, y;
int array[10][3];
int main( void )
{
    for ( x = 0; x < 3; x++ )
    for ( y = 0; y < 10; y++ )
        array[x][y] = 0;
    return 0;
}
```

6. 排错：下述代码片段有何错误？

```
int array[10];
int x = 1;

int main( void )
{
    for ( x = 1; x <= 10; x++ )
        array[x] = 99;

    return 0;
}
```

7. 编写一个程序，将随机数存储在一个包含五行四列的二维数组中，并分列打印这些值（提示：使用程序清单 8.3 介绍的 rand()函数）。
8. 使用一维数组重写程序清单 8.3。打印这 1000 个变量的平均值，然后打印各个值。请注意，别忘了每打印 10 个值后暂停。
9. 编写一个程序，对一个包含 10 个元素的数组进行初始化，其中每个元素的值等于其下标。然后，程序打印每个元素的值。
10. 修改练习 9 编写的程序。打印元素的初始值后，将这些值复制到一个新的数组中，并给每个值加 10，然后再打印新的数组。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第 9 天课程　指　针

今天的课程介绍 C 语言的一个重要组成部分——指针。指针提供了强大灵活的操纵数据的方式。今天将介绍以下内容：

- 指针的定义；
- 指针的用途；
- 如何声明和初始化指针？
- 如何使用指向变量和数组的指针？
- 如何使用指针将数组传递给函数？

阅读今天的课程时，您可能无法马上明白使用指针的优点。指针的优点包含两个方面：使用指针可以更好地完成某些工作；如果不使用指针，有些工作将无法完成。当您阅读完今天以及接下来几天的课程后，将明白这些优点的详细情况。现在，您只需记住，要成为一个精通 C 语言的程序员，必须理解指针。

9.1　指针是什么

要理解指针，您必须知道一些计算机如何在内存中存储信息的基本知识。接下来的一节简要地介绍了 PC 的内存。

9.1.1　计算机内存

PC 的内存（RAM）由数以千计（如果不是数以百万计）的顺序存储单元组成，其中每一个单元用一个唯一的地址标识。计算机的内存地址从 0 开始，最大值取决于内存量。

使用计算机时，操作系统将使用一些内存；运行程序时，程序的代码（用于完成该程序的各种任务的机器语言指令）和数据（程序使用的信息）也将使用一些内存。本节介绍程序的数据使用的内存。

在 C 程序中声明一个变量时，编译器会留出一个具有唯一的地址的内存单元来存储该变量。编译器将该地址同变量名关联起来。当程序使用该变量时，将自动访问相应的内存单元。程序使用的是内存单元的地址，而您不知道，但不必关心它。

图 9.1 说明了这一点。程序声明了一个名为 rate 的变量，并将其初始化为 100。编译器将地址为 1004 的内存单元留给了这个变量，并将地址 1004 和该变量的名称关联起来。

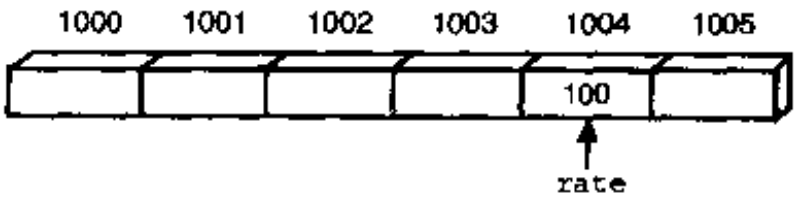


图 9.1　变量被存储在特定地址对应的内存中

9.1.2　创建指针

您应该注意到了，变量 rate 的地址是一个数字（其他变量的地址也是这样），可以像对待其他数字那样来

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

对待它。知道变量的地址后，便可以创建第二个变量，用来存储前一个变量的地址。第一步是声明一个用于保存 rate 的地址的变量，假设其名称为 p_rate。此时，p_rate 未被初始化，系统为它分配了存储空间，但它的值是不确定的，如图 9.2 所示。

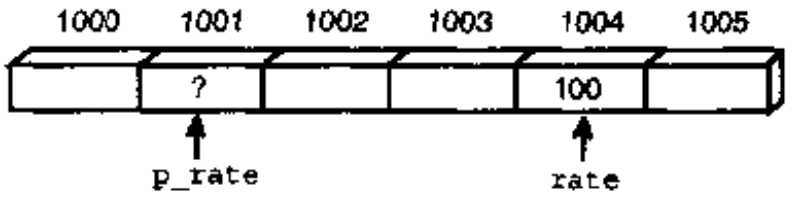


图 9.2　为变量 p_rate 分配了内存存储空间

接下来将变量 rate 的地址存储到变量 p_rate 中。由于现在 p_rate 的值为 rate 的地址，因此它指出了 rate 被存储在内存的什么位置。用 C 语言的话说，p_rate 指向 rate 或者是一个指向 rate 的指针。图 9.3 说明了这一点。

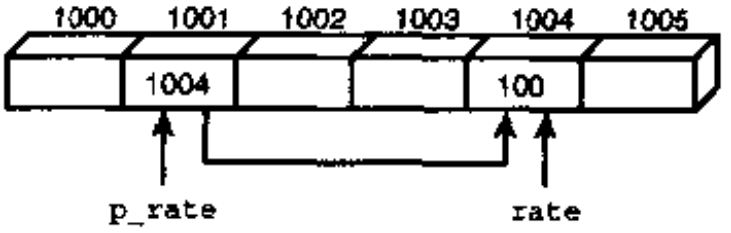


图 9.3　变量 p_rate 存储了变量 rate 的地址，因此是一个指向 rate 的指针

总之，指针是一个变量，它存储了另一个变量的地址。接下来介绍在 C 程序中使用指针的细节。

9.2　指针和简单变量

在前一个范例中，指针变量指向一个简单变量（即非数组变量）。本节介绍如何声明和使用指向简单变量的指针。

9.2.1　声明指针

指针是一个数值变量，和其他变量一样，使用指针之前必须声明。指针变量的命名规则与其他变量一样，同时也必须是唯一的。今天的课程将使用这样的约定，即将指向变量 name 的指针命名为 p_name。然而，完全可以不这样做，您可以将指针命名为任何名称，只要遵循 C 语言的命名规则即可。

指针声明的格式如下：

```
typename *ptrname;
```

其中 typename 可以是任何数据类型，它指定了指针指向的变量的类型。星号（*）是一个间接运算符，它表明 ptrname 是一个指向 typename 类型变量的指针，而不是一个类型为 typename 的变量。可以在一条语句中同时声明指针和非指针变量。下面是一些声明指针的例子：

```
char *ch1, *ch2;          /* ch1 and ch2 both are pointers to type char */
float *value, percent; /* value is a pointer to type float, and
                          /* percent is an ordinary float variable */
```



注意：符号*可用作间接运算符和乘法运算符。不必担心编译器会迷惑。*的上下文提供了足够的信息，让编译器能够知道它被用作间接运算符还是乘法运算符。

9.2.2　初始化指针

声明指针后，可以用它来做什么呢？让它指向某种东西之前，什么也做不了。与常规变量一样，您可以

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

使用未被初始化的指针，但结果是不可预测的，甚至是灾难性的。仅当存储了变量的地址后，指针才有用。地址不会无缘无故地被存储到指针中，程序必须使用地址运算符（&）将地址存储到指针中。被放置在变量名前面时，地址运算符返回该变量的地址。因此，初始化指针的格式如下：

```
pointer = &variable;
```

回到图 9.3 中的范例，将变量 p_rate 初始化为指向变量 rate 的语句如下：

```
p_rate = &rate;      /* assign the address of rate to p_rate */
```

上述语句将 rate 的地址赋给 p_rate。被初始化之前，p_rate 没有指向任何特定的东西；被初始化后，它是一个指向 rate 的指针。

9.2.3　使用指针

知道如何声明和初始化指针后，您可能想知道如何使用它们。间接运算符（*）再次找到了用武之地。当*位于指针名前面时，将引用指针指向的变量。

还是以前一个范例为例（指针已被初始化为指向变量 rate），代码*p_rate 引用的是变量 rate。如果要打印变量 rate 的值（在这个例子中，为 100），可以这样编写代码：

```
printf("%d", rate);
```

也可以这样编写代码：

```
printf("%d", *p_rate);
```

在 C 语言中，上述两条语句是等价的。使用变量名来访问变量的内容被称为直接存取；使用指向变量的指针来访问变量的内容被称为间接存取。图 9.4 说明了这样一点，即前面带间接运算符的指针名引用的是指针指向的变量的值。

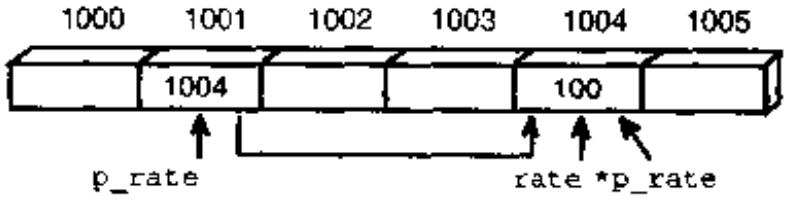


图 9.4　将间接运算符用于指针

请暂停一会儿，进一步思考一下上述内容。指针是 C 语言的有机组成部分，您必须理解它。很多人会对指针感到迷惑，因此如果您感到有些困惑，也不用担心。如果您想复习上述内容，这样做好了。也许下面的总结会对您有所帮助。

如果名为 ptr 的指针被初始化为指向变量 var，则下面的话是正确的：

- *ptr 和 var 指的是 var 的内容（即存储在 var 中的值）；
- ptr 和&var 指的是 var 的地址。

正如您看到的，不带间接运算符的指针名用来访问指针本身，即它指向的变量的地址。

程序清单 9.1 演示了指针的基本用法，您应输入、编译并运行该程序。

程序清单 9.1　　prt.c：指针的基本用法

```
1:  /* Demonstrates basic pointer use. */
2:
3:  #include <stdio.h>
4:
5:  /* Declare and initialize an int variable */
6:
7:  int var = 1;
8:
9:  /* Declare a pointer to int */
10:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn


```
11: int *ptr;
12:
13: int main( void )
14: {
15:     /* Initialize ptr to point to var */
16:
17:     ptr = &var;
18:
19:     /* Access var directly and indirectly */
20:
21:     printf("\nDirect access, var = %d", var);
22:     printf("\nIndirect access, var = %d", *ptr);
23:
24:     /* Display the address of var two ways */
25:
26:     printf("\n\nThe address of var = %d", &var);
27:     printf("\nThe address of var = %d\n", ptr);
28:
29:     return 0;
30: }
```

该程序的输入如下：

```
Direct access, var = 1
Indirect access, var = 1

The address of var = 4202504
The address of var = 4202504
```



注意：在您的系统上，输出的var的地址可能不是4202504。

分析：该程序清单声明了两个变量。第7行将var声明为int变量，并将其初始化为1。第11行声明一个名为ptr的指针，该指针用于指向int变量。第17行使用地址运算符（&）将var的地址赋给了指针ptr。程序的其他部分将这两个变量的值显示到屏幕上。第21行打印var的值，而第22行打印ptr指向的内存单元中存储的值。在这个程序中，这个值为1。第26行使用地址运算符打印var的地址，这个值与第27行使用指针变量ptr打印的值相同。

该程序清单适合用来学习，它说明了变量、变量的地址、指针和解除引用的指针之间的关系。

应　该	不应该
一定要理解指针的概念及其工作原理。要掌握C语言，必须掌握指针。	不要使用未被初始化的指针，否则结果将可能是灾难性的。

9.3　指针和变量类型

前面的讨论没有说明这样一个事实，即不同类型的变量占用的内存数量是不同的。就较常用的PC操作系统而言，short变量占用2个字节，float变量占用4个字节，等等。内存的每个字节都有地址，因此占用多个字节的变量实际上占用了多个地址。

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

那么，指针如何处理多字节变量的地址呢？其工作原理如下：变量的地址实际上是它占用的第一个字节的地址。我们通过一个声明并初始化三个变量的范例来说明这一点：

```
short vshort = 12252;
char vchar = 90;
float vfloat = 1200.156004;
```

这些变量在内存中的存储位置如图 9.5 所示，其中 short 变量占用 2 个字节，char 变量占用一个字节，float 变量占用 4 个字节。

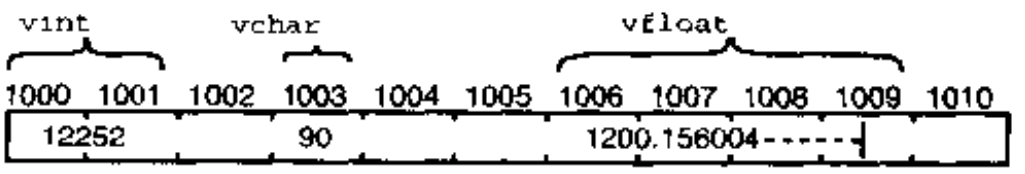


图 9.5　在内存中，不同类型的数值变量占用的存储空间不同

现在声明三个指针，并将它们初始化为指向上述三个变量：

```
int *p_vshort;
char *p_vchar;
float *p_vfloat;
/* additional code goes here */
p_vshort = &vshort;
p_vchar = &vchar;
p_vfloat = &vfloat;
```

其中每个指针的值都等于它指向的变量的第一个字节的地址。因此 p_vshort 的值为 1000，p_vchar 为 1003，p_vfloat 为 1006。然而，别忘了，每个指针都被声明为指向特定类型的变量。编译器知道，指向 short 变量的指针指向的是两个字节中的第一个字节；而类型为 float 的指针指向的是 4 个字节中的第一个字节；等等。图 9.6 说明了这一点。

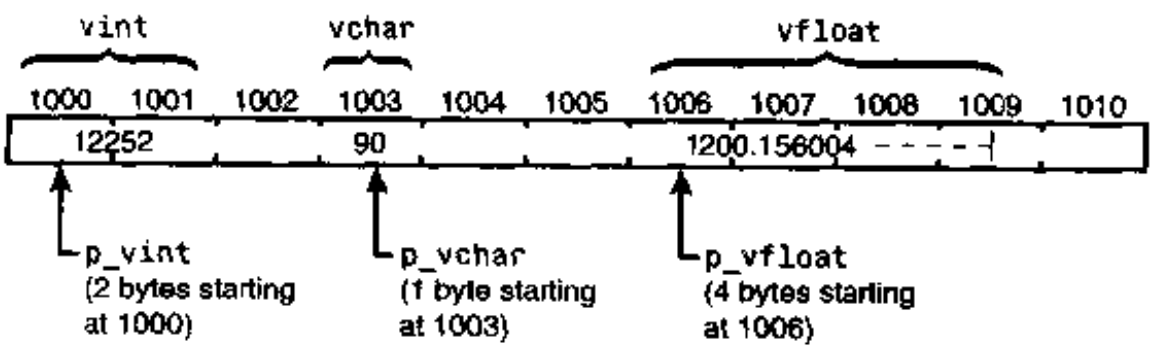


图 9.6　编译器知道指针指向的变量的长度



注意：在图 9.5 和 9.6 中，三个变量之间有一些未被使用的内存存储单元，这只是为了使图形更清晰。实际上，大多数 C 编译器将把这三个变量存储在相邻的内存单元中，它们之间不会有未被使用的字节。

9.4　指针和数组

操纵简单变量时，指针很有用；而在操纵数组时，指针更有用。在 C 语言中，指针和数组之间存在一种特殊关系。事实上，当您使用第 8 天课程介绍的数组下标表示法时，您实际上已经在使用指针，只是您不知道而已。接下来的几节将介绍其工作原理。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

9.4.1 作为指针的数组名

不带方括号的数组名是一个指针，它指向数组的第一个元素。因此，如果您声明了一个名为 data[]的数组，则 data 是第一个数组元素的地址。

您可能会问，"要获得地址，需要使用地址运算符吗？"是的。您也可以使用表达式&data[0]获得数组的第一个元素的地址。在 C 语言中，data 和&data[0]是等价的。

数组名是一个指向该数组的指针。数组名是一个指针常量，不能被修改，在整个程序运行期间，其值是固定不变的。这是有道理的：如果您修改它的值，它将指向其他地方，而不是原来的数组（该数组位于内存的某个固定位置）。

然而，您可以声明一个指针变量，并将其初始化为指向该数组。例如，下面的代码将指针变量 p_array 初始化为指向数组 array[]的第一个元素：

```
int array[100], *p_array;
/* additional code goes here */
p_array = array;
```

由于 p_array 是一个指针变量，因此可以修改它，使之指向其他地方。与数组名（array）不同，p_array 并没有被固定为指向数组 array[]的第一个元素。例如，可以修改它，使之指向 array[]的其他元素。如何完成这样的工作呢？首先您需要了解数组元素在内存中是如何存储的。

9.4.2 数组元素的存储

第 8 天的课程介绍过，数组元素在内存中是顺序存储的，第一个元素的地址最小。接下来的元素（即索引大于 0 的元素）的存储地址大些。大多少呢？这取决于数组的数据类型（char、int、float 等等）。

以类型为 short 的数组为例。第 3 天的课程介绍过，单个 short 变量占用两个字节的内存。因此每个数组元素和前一个元素相隔两个字节，因此地址比前一个元素大 2。另一方面，float 变量占用 4 个字节。在类型为 float 的数组中，每个数组元素和前一个数组元素相隔 4 个字节，因此地址比前一个元素大 4。

图 9.7 说明了在包含 6 个元素的 short 数组和包含 3 个元素的 float 数组中，数组存储空间和地址之间的关系。

```
int x[6];
```

1000 1001	1002 1003	1004 1005	1006 1007	1008 1009	1010 1011
x[0]	x[1]	x[2]	x[3]	x[4]	x[5]

```
float expenses[3];
```

1250 1251 1252 1253	1254 1255 1256 1257	1258 1259 1260 1261
expenses[0]	expenses[1]	expenses[2]

图 9.7 不同类型的数组的存储情况

从图 9.7 可知，为何下面的关系成立：

```
1: x == 1000
2: &x[0] == 1000
3: &x[1] = 1002
4: expenses == 1250
5: &expenses[0] == 1250
6: &expenses[1] == 1254
```

x 是第一个元素（x[0]）的地址。x[0]的地址为 1000，上述第二行也说明了这一点，其含义是数组 x 的第一个元素的地址为 1000。第 3 行表明，第 2 个元素（下标为 1）的地址为 1002，同样，图 9.7 说明了这一点。第 4、5、6 行几乎分别与第 1、2、3 行相同，不同的只是两个数组元素的地址之差。在类型为 short 的数组 x

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

中，地址差为 2 个字节；而在类型为 float 的数组 expenses 中，地址差为 4 个字节。

如何使用指针存取下一个数组元素呢？从这些范例可知，要存取类型为 short 的数组中的下一个元素，必须将指针加 2；而要存取类型为 float 的数组的下一个元素，必须将指针加 4。概括地说，要存储某种类型的数组的下一个元素，必须将指针加 sizeof(datatype)。第 3 天的课程介绍过，运算符 sizeof()返回数据类型的长度（单位为字节）。

程序清单 9.2 通过声明类型为 short、float、double 的数组，并显示后续元素的地址，说明了地址和数组类型之间的关系。

程序清单 9.2　　asize.c：显示相邻的数组元素的地址

```
1:  /* Demonstrates the relationship between addresses and */
2:  /* elements of arrays of different data types. */
3:
4:  #include <stdio.h>
5:
6:  /* Declare a counter and three arrays. */
7:  int ctr;
8:  short array_s[10];
9:  float array_f[10];
10: double array_d[10];
11:
12: int main( void )
13: {
14:     /* Print the table heading */
15:
16:     printf("\t\tShort\t\tFloat\t\tDouble");
17:
18:     printf("\n================================");
19:     printf("========================");
20:
21:     /* Print the addresses of each array element. */
22:
23:     for (ctr = 0; ctr < 10; ctr++)
24:        printf("\nElement %d:\t%ld\t\t%ld\t\t%ld", ctr,
25:           &array_s[ctr], &array_f[ctr], &array_d[ctr]);
26:
27:     printf("\n================================");
28:     printf("========================\n");
29:
30:     return 0;
31: }
```

该程序的输出如下：

```
                Short          Float          Double
================================================================
Element 0:      4206736        4206608        4206656
Element 1:      4206738        4206612        4206664
Element 2:      4206740        4206616        4206672
Element 3:      4206742        4206620        4206680
Element 4:      4206744        4206624        4206688
Element 5:      4206746        4206628        4206696
```

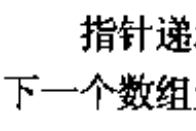

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
Element 6:      4206748         4206632         4206704
Element 7:      4206750         4206636         4206712
Element 8:      4206752         4206640         4206720
Element 9:      4206754         4206644         4206728
==========================================================
```

分析：在您的计算机上运行该程序时，显示的地址可能与上面不同，但地址之间的关系是相同的。从上述输出可知，short 数组中相邻元素的地址相差两个字节，float 数组中相邻元素的地址相差 4 个字节，double 数组则相差 8 个字节。



> 注意：在有些机器中，数据类型的长度与此不同。如果您使用的机器不同，则输出中的地址间隔可能不同，但间隔应是一致的。

该程序清单使用了第 7 天课程中介绍的转义字符。第 16 和 24 行的函数调用 printf()使用了转义字符制表符（\t）来对齐各列的内容，从面形成一个表格。

在程序清单 9.2 中，第 8、9 和 10 行声明了三个数组。第 8 行声明了一个名为 array_s 的 short 数组；第 9 行声明了一个名为 array_f 的 float 数组；第 10 行声明了一个名为 array_d 的 double 数组。第 16 行打印要显示的表格的列标题，第 18、19 和 27、28 行分别打印表格顶部和底部的短划线。对于表格而言，这是一种不错的风格。第 23～25 行是一个 for 循环，用于打印表格的各行。首先打印的是元素 ctr 的编号，然后是三个数组中相应元素的地址。

9.4.3 指针算术

有了指向数组第一个元素的指针后，要访问数组中的下一个元素，必须给指针加上一个值，即数组存储的数据类型的长度。如何使用指针表示法来访问数组元素呢？使用指针算术。

您可能会问，“我不需要再学习另一种算术吧？”请不用担心，指针算术很简单，它使得在程序中使用指针更容易。您只需关心两种运算：递增和递减。

1. 指针递增

指针递增指的是给它加上一个值。例如，当您将指针加 1 时，指针算术将自动增加指针的值，使之指向下一个数组元素。换句话说，编译器知道指针指向的数据类型（根据指针声明），并将存储在指针中的地址增加该数据类型的长度。

假设 ptr_to_short 是一个指针变量，它指向一个 short 数组的某个元素。则下面的语句：

```
ptr_to_short++;
```

将把 ptr_to_short 的值增加 short 类型的长度(通常为 2 字节)，使之指向下一个数组元素。同样，如果 ptr_to_float 指向一个 float 数组的某个元素，则下面的语句：

```
ptr_to_float++;
```

将把 ptr_to_float 的值增加 float 类型的长度（通常为 4 个字节）。

将指针增加超过 1 的值时，情况与此类似。如果将指针加 n，则编译器将该指针的值增加 n 与数据类型长度的积。因此，下面的语句：

```
ptr_to_short += 4;
```

将存储在 ptr_to_short 中的值加上 8（假设 short 的长度为 2 字节），因此该指针将指向接下来的第 4 个元素。同样，下面的语句：

```
ptr_to_float += 10;
```

将存储在 ptr_to_float 中的值加 40（假设 float 的长度为 4 字节），因此该指针将指向接下来的第 10 个元素。

2. 指针递减

指针递减的原理和指针递增相同。指针递减实际上是一种特殊的递增运算，即加上的是个负值。如果使

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

用运算符--或-=来对指针执行递减运算，指针算术将根据数组元素的长度做相应的调整。

程序清单 9.3 演示了如何使用指针算术存取数组元素。通过递增指针，程序高效地遍历数组中的所有元素。

程序清单 9.3　　ptr_math.c：使用指针算术和指针表示法存取数组元素

```
1:  /* Demonstrates using pointer arithmetic to access */
2:  /* array elements with pointer notation. */
3:
4:  #include <stdio.h>
5:  #define MAX 10
6:
7:  /* Declare and initialize an integer array. */
8:
9:  int i_array[MAX] = { 0,1,2,3,4,5,6,7,8,9 };
10:
11: /* Declare a pointer to int and an int variable. */
12:
13: int *i_ptr, count;
14:
15: /* Declare and initialize a float array. */
16:
17: float f_array[MAX] = { .0, .1, .2, .3, .4, .5, .6, .7, .8, .9 };
18:
19: /* Declare a pointer to float. */
20:
21: float *f_ptr;
22:
23: int main( void )
24: {
25:     /* Initialize the pointers. */
26:
27:     i_ptr = i_array;
28:     f_ptr = f_array;
29:
30:     /* Print the array elements. */
31:
32:     for (count = 0; count < MAX; count++)
33:         printf("%d\t%f\n", *i_ptr++, *f_ptr++);
34:
35:     return 0;
36: }
```

该程序的输出如下：

```
0       0.000000
1       0.100000
2       0.200000
3       0.300000
4       0.400000
5       0.500000
6       0.600000
7       0.700000
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
8       0.800000
9       0.900000
```

分析：在该程序中，第 5 行定义了一个名为 MAX 的常量，并将其设置为 10。第 9 行使用 MAX 指定名为 i_array 的 int 数组包含的元素数目。在声明该数组的同时对其元素进行了初始化。第 13 行声明了另外两个 int 变量。第一个是一个名为 i_ptr 的指针，您之所以知道这是一个指针是因为它前面有间接运算符（*）；另一个是一个简单的 int 变量，名为 count。第 17 行声明并初始化了另一个数组，该数组的类型为 float，包含 MAX 个元素。第 21 行声明一个指向 float 变量的指针，名为 f_ptr。

main()函数位于第 23～36 行。在第 27 和 28 行，程序将这两个数组的开始地址分别赋给相应类型的指针。别忘了，不带下标的数组名指的是数组的开始地址。第 32～33 行的 for 循环使用 int 变量 count 来计数从 0 到 MAX 的值，每次循环时，第 33 行都在函数调用 printf()中解除两个指针的引用，并打印它们的值。然后使用递增运算符将每个指针递增，使它们指向数组的下一个元素，然后进行 for 循环的下一次迭代。

您可能认为，不使用指针，而使用数组下标表示法也无妨。情况确实如此，对于像这样的简单编程任务，指针表示法并没有什么优势可言。然而，当您编写更复杂的程序时，指针的优势将显现出来。

别忘了，不能对指针常量执行递增或递减运算（不带方括号的数组名便是一个指针常量）；另外，当您操纵指向数组元素的指针时，编译器不会跟踪数组的开始和结束位置。如果您不小心对指针执行递增或递减运算，则可能使之指向数组的前面或后面。而这里可能存储了数据，但不是数组元素。因此，您必须注意指针指向的位置）。

3. 其他指针操作

您可能想使用的另一种指针算术运算是求差，即将两个指针相减。如果两个指针指向同一个数组的不同元素，可以通过将它们相减来获知它们之间的距离。同样，指针算术也将自动调整，使结果为元素数。因此，如果 ptr1 和 ptr2 指向同一个数组（可以是任何类型）的不同元素，则下面的表达式将告诉您这两个元素之间的距离：

```
ptr1 - ptr2;
```

也可以对指针进行比较。仅当两个指针指向同一个数组时，对它们进行比较才是合法的。在这种情况下，关系运算符==、!=、>、<、>=和<=都能正确地运行。前面的数组元素（即下标较小）的地址总是小于后面的元素。因此，当 ptr1 和 ptr2 指向同一个数组的元素时，如果 ptr1 指向的元素在 ptr2 指向的元素的前面，则下面的表达式为真：

```
ptr1 < ptr2;
```

至此介绍了所有允许的指针运算。很多可用于常规变量的算术运算，如乘法和除法，对于指针来说是没有意义的。C 编译器不允许这样的运算，例如如果 ptr 是一个指针，则下面的语句将导致错误：

```
ptr *= 2;
```

表 9.1 列出了可以对指针执行的所有运算，这些运算都在今天的课程中介绍过了。

表 9.1　　指针运算

运　算	描　述
赋值	可以将一个值赋给指针。这个值必须是使用地址运算符（&）获得的地址或来自指针常量（数组名）的地址。
间接运算	间接运算符（*）提供存储在指针指向的位置的值（通常被称为解除引用）。
求地址	可以使用地址运算符获得指针的地址，因此指针可以指向另一个指针。这是一个高级主题，将在第 15 天的课程中介绍。
递增	可以给指针加上一个整数，使之指向另一个内存单元。
递减	可以将指针减去一个整数，使之指向另一个内存单元。
求差	可以将两个指针相减，从而获知他们之间的距离。
比较	只能对指向同一个数组的指针进行比较。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

9.5　有关指针的注意事项

编写使用指针的程序时，必须避免一种严重的错误：不要在赋值语句的左边使用未被初始化的指针。例如，下面的语句声明了一个 int 类型的指针：

```
int *ptr;
```

该指针未被初始化，因此它不指向任何东西。更准确地说，它没有指向任何已知的东西。未被初始化的指针有值，只是您不知道是什么。在很多情况下，它为 0。如果您在赋值语句中使用未被初始化的指针，如：

```
*ptr = 12;
```

则 12 将被赋给 ptr 指向的地址。该地址可能是内存的任何地方——可能是存储操作系统代码或程序代码的地方。将 12 存储到这个位置可能会覆盖一些重要的信息，这可能导致奇怪的程序错误，甚至整个系统崩溃。

对于使用未被初始化的指针而言，赋值语句的左边是最为危险的地方。在程序的其他地方使用未被初始化的指针时，也会导致其他错误，虽然这些错误没有那么严重，因此使用指针之前，一定要对它进行合适的初始化。您必须自己完成这种工作，不要认为编译器为您完成它们。

应　该	不应该
请切记，给指针加上或减去一个整数时，将根据它指向的数据类型来修改指针的值，而不是将其值加（减）1 或指定的整数（除非该指针指向的是一个字节的字符变量）。 一定要了解在您的计算机上，变量类型的长度，操纵指针和内存时，您必须知道变量的长度	不要对指针执行诸如乘法、除法或求模等数学运算，对指针执行加（递增）减（递减）运算是允许的。 不要对数组变量执行递增或递减运算，而应将数组的开始位置赋给一个指针，然后对该指针执行递增或递减运算（参见程序清单 9.3）。

9.6　数组下标表示法和指针

不带方括号的数组名是一个指针，指向数组的第一个元素。因此，可以使用间接运算符存取第一个数组元素。如果声明了一个名为 array[]的数组，则表达式*array 指的是第一个数组元素，而*(array + 1)是数组的第二个元素，依此类推。就整个元素而言，下面的关系成立：

```
*(array) == array[0]
*(array + 1) == array[1]
*(array + 2) == array[2]
...
*(array + n) == array[n]
```

这说明了数组下标表示法和数组指针表示法之间的等价关系。在程序中，您可以使用其中任何一种表示法，C 编译器将它们视为两种使用指针存取数组数据的不同方式。

9.7　将数组传递给函数

今天的课程讨论了指针和数组之间的特殊关系，当您需要把数组作为参数传递给函数时，这种关系将发挥作用。您只能使用指针来将数组传递给函数。

第 5 天介绍过，参数是调用程序传递给函数的值，其类型可以是 int、float 或其他任何简单数据类型，但必须是单个的数值。它可以是单个的数组元素，但不能是整个数组。如果需要将整个数组传递给函数，该如何办呢？您可以声明一个指向该数组的指针，而且该指针是单个的数值（数组第一个元素的地址）。如果您将这个值传递给函数，函数将知道数组的地址，从而可以使用指针表示法来存取该数组的元素。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

接下来考虑另一个问题。编写以数组作为参数的函数时，您希望它能够处理长度不同的数组。例如，您可以编写一个这样的函数：找出整型数组中最大的元素。如果它只能处理长度固定的数组，则用处不大。

函数如何知道地址被传递给它的数组的长度呢？别忘了，传递给函数的是一个指针，该指针指向数组的第一个元素，这可能是 10 个或 1000 个元素中的第一个元素。有两种让函数能够知道数组长度的方法。

可以在最后一个元素中存储特殊的值来标识它。函数处理数组时，将在每个元素中查找这样的值，找到这个值后，便说明到达了数组的末尾。这种方法的缺点是，您必须保留一个值，将其作为数组末尾的指示符，因此降低了在数组中存储实际数据的灵活性。

另一种更灵活、更简单的方法是，将数组长度作为一个参数传递给函数，本书将使用这种方法。该参数的类型为 int，因此需要给函数传递两个参数：指向第一个元素的指针和指定数组中元素数目的整数。

程序清单 9.4 让用户输入一系列的值，并将它们存储在一个数组中。然后调用函数 largest()，并将上述数组（包括指针和长度）传递给该函数。该函数找到数组中最大的值，并将其返回给调用它的程序。

程序清单 9.4　　passing.c：将数组传递给函数

```
1:  /* Passing an array to a function. */
2:
3:  #include <stdio.h>
4:
5:  #define MAX 10
6:
7:  int array[MAX], count;
8:
9:  int largest(int num_array[], int length);
10:
11: int main( void )
12: {
13:     /* Input MAX values from the keyboard. */
14:
15:     for (count = 0; count < MAX; count++)
16:     {
17:         printf("Enter an integer value: ");
18:         scanf("%d", &array[count]);
19:     }
20:
21:     /* Call the function and display the return value. */
22:     printf("\n\nLargest value = %d\n", largest(array, MAX));
23:
24:     return 0;
25: }
26: /* Function largest() returns the largest value */
27: /* in an integer array */
28:
29: int largest(int num_array[], int length)
30: {
31:     int count, biggest = -12000;
32:
33:     for ( count = 0; count < length; count++)
34:     {
35:         if (num_array[count] > biggest)
36:             biggest = num_array[count];
37:     }
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
38:
39:     return biggest;
40: }
```

该程序的运行情况如下：

```
Enter an integer value: 1
Enter an integer value: 2
Enter an integer value: 3
Enter an integer value: 4
Enter an integer value: 5
Enter an integer value: 10
Enter an integer value: 9
Enter an integer value: 8
Enter an integer value: 7
Enter an integer value: 6
Largest value = 10
```

分析：该程序使用的函数名为 largest()，它接受一个指向数组的指针。该函数的原型位于第 9 行，它与第 29 行的函数头相同，只是多了一个分号。

对于第 29 行的函数头中的大部分内容，您应该耳熟能详：函数 largest()返回一个 int 值；其第二个参数名为 length，类型为 int。唯一的新内容 int num_array[]，它表明第一个参数是一个 int 指针，名为 num_array。也可以这样编写函数头和函数声明：

```
int largest(int *num_array, int length);
```

这与前一种格式等价，int num_array[]和 int *num_array 指的都是指向 int 变量的指针。第一种格式可能更好，因为它提醒您，该参数是一个指向数组的指针。当然，该指针并不知道它指向的是一个数组，但函数是以这种方式使用该指针的。

下面来看看函数 largest()。当该函数被调用时，形参 num_array 将存储第一个实参的值，即指向数组第一个元素的指针。在 largest()函数中，第 35 和 36 行使用下标表示法来存取数组元素。您也可以使用指针表示法，将 if 语句重写为如下所示：

```
for (count = 0; count < length; count++)
{
      if (*(num_array+count) > biggest)
        biggest = *(num_array+count);
}
```

程序清单 9.5 演示了另一种将数组传递给函数的方式。

程序清单 9.5　　passing2.c：另一种将数组传递给函数的方式

```
1:  /* Passing an array to a function. Alternative way. */
2:
3:  #include <stdio.h>
4:
5:  #define MAX 10
6:
7:  int array[MAX+1], count;
8:
9:  int largest(int num_array[]);
10:
11: int main( void )
12: {
13:     /* Input MAX values from the keyboard. */
14:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
15:     for (count = 0; count < MAX; count++)
16:     {
17:         printf("Enter an integer value: ");
18:         scanf("%d", &array[count]);
19:
20:         if ( array[count] == 0 )
21:             count = MAX;              /* will exit for loop */
22:     }
23:     array[MAX] = 0;
24:
25:     /* Call the function and display the return value. */
26:     printf("\n\nLargest value = %d\n", largest(array));
27:
28:     return 0;
29: }
30: /* Function largest() returns the largest value */
31: /* in an integer array */
32:
33: int largest(int num_array[])
34: {
35:     int count, biggest = -12000;
36:
37:     for ( count = 0; num_array[count] != 0; count++)
38:     {
39:         if (num_array[count] > biggest)
40:             biggest = num_array[count];
41:     }
42:
43:     return biggest;
44: }
```

该程序的运行情况如下：

```
Enter an integer value: 1
Enter an integer value: 2
Enter an integer value: 3
Enter an integer value: 4
Enter an integer value: 5
Enter an integer value: 10
Enter an integer value: 9
Enter an integer value: 8
Enter an integer value: 7
Enter an integer value: 6

Largest value = 10
```

下面是再次运行该程序的情况：

```
Enter an integer value: 10
Enter an integer value: 20
Enter an integer value: 55
Enter an integer value: 3
Enter an integer value: 12
Enter an integer value: 0
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
Largest value = 55
```

分析：该程序中的 largest()函数的功能与程序清单 9.4 完全相同。区别在于需要使用数组标记。第 37 行的 for 循环不断查找最大的值，直到元素的值为 0（这表明达到了数组的末尾）。

从程序前面的部分可以知道程序清单 9.5 和 9.4 之间的区别。首先，第 7 行在数组中增加了一个元素，该元素用于存储一个指示数组末尾的值。在第 20 和 21 行增加了一条 if 语句，用于检查用户输入的是否为 0，如果是，则表明用户要结束输入。当用户输入 0 后，count 将被设置为 MAX，从而退出 for 循环。第 23 行确保当用户输入了 MAX 值时，最后一个元素为 0。

通过添加额外的命令来处理数据输入，可以使用 largest()函数能够处理任意长度的数组，然而，这是一把双刃剑。如果您忘记在数组的最后放置 0，情况将如何呢？largest()将跨越数组的末尾，继续比较内存中的值，直到找到 0 为止。

正如您看到的，将数组传递给函数并不太难。您只需传递一个指向数组第一个元素的指针即可。在大多数情况下，还需要传递数组包含的元素数目。在函数中，可以使用该指针以下标表示法或指针表示法来存取数组元素。



> 警告：第 5 天的课程介绍过，将简单变量传递给函数时，传递的只是该变量的值的副本。函数可以使用这个值，但不能修改原来的变量，因为它无法访问该变量。将数组传递给函数时，情况完全不同。传递给函数的是数组的地址，而不是数组中的值的副本。函数中的代码操纵的是实际的数组元素，因此能够修改存储在数组中的值。

9.8　总　结

今天的课程介绍了指针——C 语言编程的核心。指针是一个变量，它存储的是另一个变量的地址；因此人们说，指针指向它存储的地址所对应的变量。与指针相关的运算符有两个：地址运算符（&）和间接运算符（*）。被放置在变量的前面时，地址运算符返回该变量的地址；被放置在指针的前面时，间接运算符返回该指针指向的变量的内容。

指针和数组之间存在一种特殊关系。不带方括号的数组名是一个指针，它指向数组的第一个元素。指针算术的特征使得使用指针存取数组元素很容易。数组下标表示法实际上是指针表示法的一种特殊形式。

您还学习了如何通过传递指向数组的指针来将数组传递给函数。函数知道数组的地址和长度后，便可以使用指针表示法或下标表示法来存取数组元素。

9.9　问与答

问：在 C 语言中，指针为何如此重要？

答：指针让您能够更好地控制计算机和数据。和函数一起使用时，指针使您能够修改被传递给函数的变量。第 15 天的课程将介绍指针的其他用途。

问：编译器如何知道*是被用作乘法运算符、间接运算符还是被用来声明指针？

答：编译器根据上下文来解释星号的用途。如果*所在的语句以变量类型打头，则认为它被用来声明一个指针；如果被用于一个已经声明过的指针，但不位于变量声明中，则被认为是间接运算符；如果被用于数学表达式中，且右边不是指针变量，则被认为是一个乘法运算符。

问：如果对指针使用地址运算符，情况将如何？

答：您将得到指针的地址。别忘了，指针也是一个变量，存储的是它指向的变量的地址。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

问：变量总是被存储在同一个位置吗？

答：不是。每次程序被运行时，变量可能存储在不同的地址中。决不要将一个地址常量赋给指针。

9.10 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

9.10.1 小测验

1. 哪个运算符被用来获得变量的地址？
2. 哪个运算符被用来获得指针指向的内存单元的值？
3. 指针是什么？
4. 解除引用指的是什么？
5. 数组的元素在内存中是如何存储的？
6. 指出两种获得数组 data[]的第一个元素的地址的方式。
7. 将数组传递给函数时，函数通过哪两种方式来确定是否已到数组的末尾？
8. 今天的课程介绍了哪 6 种可用于指针的运算？
9. 假设有两个指针，其中第一个指针指向一个类型为 int 的数组的第三个元素，而第二个指针指向该数组的第四个元素，则第二个指针和第一个指针的差是多少（假设 int 类型的长度为 2 个字节）？
10. 假设问题 9 中的数组类型为 float，则这两个指针的差为多少（假设 float 类型的长度为 4 个字节）？

9.10.2 练习

1. 声明一个名为 char_ptr、类型为 char 的指针。
2. 如果有一个名为 cost、类型为 int 的变量，请声明一个名为 p_cost 的指针，并将它初始化为指向该变量。
3. 对于练习 2 中的情况，分别使用直接存取和间接存取的方式将 100 赋给变量 cost。
4. 对于练习 3 中的情况，打印指针的值和它指向的变量的值。
5. 将 float 变量 radius 的地址赋给一个指针。
6. 使用两种不同的方式，分别将 100 赋给数组 data[]的第三个元素。
7. 编写一个名为 sumarrays()的函数，它接受两个数组作为参数，将两个数组中的所有值相加，并返回得到的结果。
8. 在一个简单的程序中使用在练习 7 中编写的函数。
9. 编写一个名为 addarrays()的函数，它接受两个长度相同的数组，将这两个数组中对应的元素相加，并将结果放到第三个元素中。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

TYPE & RUN 3　让程序暂停

这是第三个 Type & Run。别忘了，Type & Run 中的程序清单旨在提供一些功能比课程中的程序强些的程序。该程序清单只包含几项还未介绍过的内容，以使其易于理解。输入并运行该程序后，花一些时间对其中的代码进行试验。对程序进行修改，然后重新编译并运行它们，看看会出现什么情况。如果出现错误，请确保您正确地输入了该程序清单。

程序清单 T&R 3　　**seconds.c：包含一个休眠函数的程序**

```
1:  /* seconds.c */
2:  /* Program that pauses. */
3:
4:  #include <stdio.h>
5:  #include <stdlib.h>
6:  #include <time.h>
7:
8:  void sleep( int nbr_seconds );
9:
10:  int main( void )
11: {
12:     int ctr;
13:     int wait = 13;
14:
15:     /* Pause for a number of seconds. Print a  *
16:      * dot each second waited.                 */
17:
18:     printf("Delay for %d seconds\n", wait );
19:     printf(">");
20:
21:     for (ctr=1; ctr <= wait; ctr++)
22:     {
23:       printf(".");      /* print a dot */
24:       fflush(stdout);   /* force dot to print on buffered machines */
25:       sleep( (int) 1 );  /* pause 1 second */
26:     }
27:     printf( "Done!\n");
28:     return (0);
29: }
30:
31: /* Pauses for a specified number of seconds */
32: void sleep( int nbr_seccnds )
33: {
34:     clock_t goal;
```

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
35:
36:     goal = ( nbr_seconds * CLOCKS_PER_SEC ) + clock();
37:
38:     while( goal > clock() )
39:     {
40:        ; /* loop */
41:     }
42: }
```

该程序包含一个在您编写其他程序时可能很有用的函数。函数 sleep()让程序暂停一段时间。在此期间，程序只是检查是否等待了足够长的时间。指定的时间过去后，该函数将把控制权返回给调用程序。该函数（及其变体）的用途很多。由于计算机的速度非常快，您常常想让程序暂停，让读者有充足的时间来阅读屏幕上的信息。例如，您可能想在程序开始运行时，显示一个版权屏幕。

作为一种简单的演示，该程序的主要部分打印句点。每打印一个句点，程序使用前面介绍的 sleep()函数来暂停一段时间。为增加趣味性，您可以增长暂停的时间，并使用秒表来确定计算机暂停时间的准确性。

您也可以对该程序清单进行修改，使之在暂停期间打印“x"（或其他值），方法是将第 40 行修改为如下所示：

```
printf("x");
```



警告：如果您的操作系统中有一个名为 sleep 的程序，您可能需要将该程序改为其他的名称，以免发生冲突。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 10 天课程　字符和字符串

字符是一个字母、数字、标点或其他诸如此类的符号；字符串是任何字符序列。字符串用于存储由字母、数字、标点和其他符号组成的文本数据。在编程中，字符和字符串很有用。今天将介绍以下内容：

- 如何使用 char 数据类型存储单个字符？
- 如何声明 char 类型的数组来存储由多个字符组成的字符串？
- 如何初始化字符和字符串？
- 如何使用指向字符串的指针？
- 如何打印和输入字符和字符串？

10.1　char 数据类型

C 语言使用 char 数据类型来存储字符。第 3 天的课程介绍过，char 是一种整型数值类型。既然 char 是一种数值类型，它为何可用来存储字符呢？

原因在于 C 语言存储字符的方式。在计算机内存中，所有的数据都是以数值方式存储的。字符并不能直接存储，但每个字符都有对应的数值编码。这种编码被称为 ASCII 码或 ASCII 字符集（ASCII 表示美国信息交换标准码）。在这种编码中，每个大小写字母、数字、标点和其他符号都对应于一个 0～255 的值。附录 A 列出了 ASCII 字符集。



注意：ASCII 码和 ASCII 字符集只能用于使用单字节字符集的系统；在使用多字节字符集的系统中，您将使用另一种字符集。这些内容超出了本书的范围。

例如，字母 a 的 ASCII 码为 97。在 char 变量中存储字符 a 时，实际存储的是 97。由于 char 类型的取值范围对应于标准的 ASCII 字符集，因此 char 类型非常适合用于存储字符。

此时您可能有些困惑。既然 C 语言将字符存储为数字，程序如何知道 char 变量存储的是一个字符还是数字？后面将介绍，仅仅将变量声明为 char 类型还不够，还需要对该变量执行其他操作：

- 如果 char 变量被用在程序期望得到一个字符的地方，则被解释为字符；
- 如果 char 变量被用在程序期望得到一个数字的地方，则被解释为数字。

现在，您知道 C 语言如何使用数值数据类型来存储字符数据了，接下来介绍有关字符的细节。

10.2　使用字符变量

与其他变量一样，使用 char 变量之前必须声明，另外也可以在声明的同时对它进行初始化。下面是一些范例：

```
char a, b, c;          /* Declare three uninitialized char variables */
char code = 'x';       /* Declare the char variable named code*/
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
                    /* and store the character x there */
code = '!';         /* Store ! in the variable named code*/
```

要创建字面字符常量，可以使用单引号将一个字符括起。编译器自动将字面字符常量转换为相应的 ASCII 码。可以使用编译指令#define 或关键字 const 来创建符号字符常量：

```
#define EX 'x'
char code = EX;     /* Sets code equal to 'x' */
const char A = 'Z';
```

知道如何声明和初始化字符变量后，我们来看一个这样的例子。程序清单 10.1 使用第 7 天介绍的 printf() 函数说明了字符变量的数值特性。函数 printf()可用于打印字符和数字。格式字符串%c 指示 printf()打印一个字符，而%d 指示它打印一个十进制整数。程序清单 10.1 初始化了两个 char 变量，并将它们作为字符和数字打印出来。

程序清单 10.1　　chars.c：char 变量的数值特性

```
1:  /* Demonstrates the numeric nature of char variables */
2:
3:  #include <stdio.h>
4:
5:  /* Declare and initialize two char variables */
6:
7:  char c1 = 'a';
8:  char c2 = 90;
9:
10: int main( void )
11: {
12:     /* Print variable c1 as a character, then as a number */
13:
14:     printf("\nAs a character, variable c1 is %c", c1);
15:     printf("\nAs a number, variable c1 is %d", c1);
16:
17:     /* Do the same for variable c2 */
18:
19:     printf("\nAs a character, variable c2 is %c", c2);
20:     printf("\nAs a number, variable c2 is %d\n", c2);
21:
22:     return 0;
23: }
```

该程序的输出如下：

```
As a character, variable c1 is a
As a number, variable c1 is 97
As a character, variable c2 is Z
As a number, variable c2 is 90
```

分析：第 3 天的课程介绍过，char 变量的最大取值为 127，而最大的 ASCII 码为 255。ASCII 码通常被分为两部分。标准 ASCII 码的最大值为 127，这包括所有的字母、数字、标点和其他键盘符号。编码 128～255 是扩展 ASCII 码，表示的是特殊字符，如外来字符和图形符号（完整的列表，请参阅附录 A）。因此，对于标准文本数据，可以使用 char 变量来存储；如果要打印扩展 ASCII 字符，则必须使用 unsigned char 变量。

程序清单 10.2 打印一些扩展 ASCII 字符。

程序清单 10.2　　ascii.c：打印扩展 ASCII 字符

```
1:  /* Demonstrates printing extended ASCII characters */
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
2:
3:  #include <stdio.h>
4:
5:  unsigned char mychar;    /* Must be unsigned for extended ASCII */
6:
7:  int main( void )
8:  {
9:      /* Print extended ASCII characters 180 through 203 */
10:
11:     for (mychar = 180; mychar < 204; mychar++)
12:     {
13:         printf("ASCII code %d is character %c\n", mychar, mychar);
14:     }
15:
16:     return 0;
17: }
```

该程序的输出如下：

```
ASCII code 180 is character ´
ASCII code 181 is character µ
ASCII code 182 is character ¶
ASCII code 183 is character ·
ASCII code 184 is character ¸
ASCII code 185 is character ¹
ASCII code 186 is character º
ASCII code 187 is character »
ASCII code 188 is character 1/4
ASCII code 189 is character 1/2
ASCII code 190 is character 3/4
ASCII code 191 is characteri ¿
ASCII code 192 is character À
ASCII code 193 is character Á
ASCII code 194 is character Â
ASCII code 195 is character Ã
ASCII code 196 is character Ä
ASCII code 197 is character Å
ASCII code 198 is character /E
ASCII code 199 is character Ç
ASCII code 200 is character È
ASCII code 201 is character É
ASCII code 202 is character Ê
ASCII code 203 is character Ë
```

分析：该程序的第 5 行声明了一个无符号的字符变量 mychar，这使得变量的取值范围为 0～255。与其他数值数据类型一样，不能将 char 变量初始化为一个不允许的值，否则结果将出乎意料。在第 11 行，mychar 被初始化为 180，没有超出取值范围。在 for 循环中，mychar 被逐渐递增，直到为 204。mchar 每递增一次，第 13 行都打印 mychar 的值以及对应的字符。请记住，%c 用于打印 mychar 的值对应的 ASCII 字符。

应　该	不应该
打印数字对应的字符时，一定要使用%c。	初始化字符变量时，不要使用双引号。
初始化 char 变量时，一定要使用单引号。	不要将扩展 ASCII 字符存储到有符号字符变量中。
请查看附录 A 中的 ASCII 表，了解一些可以打印的有趣的字符。	

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com



警告：有些计算机系统可能使用其他字符集，但在大多数系统中，ASCII 值 0～127 对应的字符是相同的。

10.3 使用字符串

char 变量只能存储一个字符，因此其用途有限。您需要一种存储字符串的方式。字符串是一个字符序列，人的姓名和住址都是字符串。虽然没有专门用来存储字符串的数据类型，但 C 语言使用字符数组来处理这类信息。

10.3.1 字符数组

例如，要存储包含 6 个字符的字符串，可以声明一个包含 7 个元素的 char 数组。声明字符数组的方式和其他数据类型的数组相同。例如，下面的语句：

```
char string[10];
```

声明一个包含 10 个元素的字符数组，该数组可用于存储包含 9 个或九个以下字符的字符串。

您可能会问："该数组有 10 个元素，为何只能存储 9 个字符？"在 C 语言中，字符串被定义为以空字符结尾的字符序列。空字符是一个特殊的字符，用\0 表示。虽然它是用两个字符（反斜杠和 0）表示的，但实际上是一个字符，其 ASCII 码为 0。\0 是 C 语言中的一个转义序列。



注意：有关转义序列，请参阅第 7 天的课程。

例如，当 C 程序存储字符串 Alabama 时，将总共存储 8 个字符——该字符串中的 7 个字符，加上空字符\0。因此字符数组能够存储的最长的字符串包含的字符数比数组的元素数目少 1。

10.3.2 初始化字符数组

与其他数据类型一样，也可以在声明字符数组的同时对其进行初始化。可以逐个给字符数组的元素赋值，如下所示：

```
char string[10] = { 'A', 'l', 'a', 'b', 'a', 'm', 'a', '\0' };
```

然而，更方便的方式是，使用一个字面字符串——用双引号括起的字符序列：

```
char string[10] = "Alabama";
```

当您在程序中使用字面字符串时，编译器自动在该字符串的末尾加上一个空字符。如果您声明数组时，没有指定元素数目，编译器将替您计算数组的长度。因此，下面的代码行声明并初始化一个包含 8 个元素的数组：

```
char string[] = "Alabama";
```

请记住，字符串必须以空字符结尾。操纵字符串的函数（将在第 17 天的课程中介绍）通过查找空字符来确定字符串的长度。这些函数没有确定字符串末尾的其他方式，如果您遗漏了空字符，程序会认为该字符串一直延续到内存中的下一个空字符。这类错误将导致非常令人讨厌的 bug。

10.4 字符串和指针

字符串被存储在字符数组中，并用空字符来标记末尾。由于标记了字符串的末尾，因此要指定某个字符串，只需指定一个指向该字符串开头的指针即可。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 9 天的课程介绍过，数组名是一个指针，它指向该数组的第一个元素。因此，对于存储在数组中的字符串，要访问它，只需知道该数组的名称即可。事实上，在 C 语言中，访问字符串的标准方法便是使用数组名。

更准确的说，C 语言中的库函数期望使用数组名来访问字符串。C 标准库中包含大量用于操纵字符串的函数（这些函数将在第 17 天的课程中介绍）。要将字符串传递给这样的函数，只需传递数组名即可。字符串显示函数 printf()和 puts()的情况也是如此，这些函数将在今天课程的后面介绍。

您可能注意到了，前面说的是“存储在数组中的字符串”。这是否意味着有些字符串不是存储在数组中呢？确实是这样，下一节将解释原因。

10.5　不存储在数组中的字符串

前一节介绍过，字符串是由字符数组的名称和空字符定义的。字符数组的名称是一个 char 指针，它指向字符串的开头；而空字符标记了字符的末尾。数组中的字符串实际所在的位置并不重要。事实上，数组唯一的用途是为字符串提供分配的空间。

如果不通过分配数组也可以获得内存空存空间，则可以将字符串和空字符存储到那里，并使用一个指向第一个字符的指针来标记字符串的开头，就像字符串被存储在一个数组中一样。如何获得内存空间呢？有两种方法：一是在程序编译时，为字面字符串分配空间；另一种方法是在程序执行时，使用 malloc()来分配空间，这被称为动态分配。

10.5.1　编译时分配字符空间

正如前面指出的，字符串的开头是由一个 char 指针来指示的。您可能还记得如何声明这种指针：

```
char *message;
```

上述语句声明了一个名为 message 的 char 指针。现在，它不指向任何东西，但如果将上述指针声明修改为如下所示，情况将如何呢？

```
char *message = "Great Caesar\'s Ghost!";
```

当上述语句执行时，字符串 Great Caesar's Ghost!（和一个结尾的空字符）将被存储在内存的某个地方，而指针 message 将被初始化为指向该字符串的第一个字符。请不用担心该字符串到底被存储在内存的什么地方，这是由编译器自动处理的。一旦被定义，message 便是一个指向该字符串的指针，并可以这样使用它。

前面的声明/初始化语句与下面的语句等价，另外表示法*message 和 message[]也等价，它们指的都是“指针”。

```
char message[] = "Great Caesar\'s Ghost!";
```

如果编写程序时，知道需要多少内存空间，则这种为字符串分配内存空间的方式是可行的。但如果程序需要的内存空间将随用户的输入或编写程序时还不知道的因素而异，该如何办呢？可以使用 malloc()，它使您能够在程序运行时动态地分配内存空间。

10.5.2　malloc()函数

malloc()是 C 语言中的一个内存分配函数。调用该函数时，您将需要分配的字节数传递给它。malloc()将找到并保留一个所需大小的内存块，并返回其第一个字节的地址。您无需关心该内存块的位置，这是自动处理的。

malloc()返回一个地址，该函数的返回类型为 void 指针。为何是 void 指针？这种类型的指针能与所有的数据类型兼容。由于 malloc()分配的内存可能用来存储任何数据类型，因此 void 返回类型是合适的。

malloc()函数的语法如下：

```
#include <stdlib.h>
void *malloc(size_t size);
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

malloc()分配一个大小为 size 的内存块。通过使用 malloc()根据需要来分配内存，而不是在程序启动时分配，可以更高效地使用内存。使用 malloc()时，必须包含头文件 STDLIB.H。对于有些编译器，可以包含其他头文件，但出于移植性的考虑，最好包含 stdlib.h。

malloc()返回一个指向分配的内存的指针。如果 malloc()无法分配所需数量的内存，则返回 null。每当您试图分配内存时，都应检查返回的值，即使要分配的内存量很小。

下面是三个使用 malloc()的例子：

范例 1：

```
#include <stdlib.h>
#include <stdio.h>
int main( void )
{
      /* allocate memory for a 100-character string */
      char *str;
      str = (char *) malloc(100);
      if (str == NULL)
      {
            printf( "Not enough memory to allocate buffer\n");
            exit(1);
      }
      printf( "String was allocated!\n" );
      return 0;
}
```

范例 2：

```
/* allocate memory for an array of 50 integers */
int *numbers;
numbers = (int *) malloc(50 * sizeof(int));
```

范例 3：

```
/* allocate memory for an array of 10 float values */
float *numbers;
numbers = (float *) malloc(10 * sizeof(float));
```

10.5.3 使用 malloc()函数

可以使用 malloc()来分配用于存储一个字符的内存。首先声明一个 char 指针：

```
char *ptr;
```

接下来调用 malloc()，并将所需内存的大小传递给它。由于一个字符通常占用一个字节，因此所需内存的大小为一字节。然后，将 malloc()返回的值赋给指针：

```
ptr = malloc(1);
```

上述语句分配一个 1 字节的内存块，并将其地址赋给 ptr。与程序中声明的其他变量不同，这一字节的内存没有名称，而只能使用该指针来引用它。例如，要将字符 x 存储到这里，可以这样编写代码：

```
*ptr = 'x';
```

使用 malloc()为字符串分配存储空间与为单个字符分配存储空间几乎相同。主要区别在于，您需要知道要分配多少空间——字符串中包含的最大字符数。这个最大值取决于程序的需要。例如，假设要为一个包含 99 个字符的字符串（加上一个空字符，总共为 100 个字符）分配内存，则首先需要声明一个 char 指针，然后调用 malloc()：

```
char *ptr;
ptr = malloc(100);
```

现在，ptr 指向的是一个预留的 100 字节的内存块，该内存块可用来存储字符串，并可以操纵它。可以像程序已经使用下面的数组声明来显式地分配内存那样，来使用 ptr：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
char ptr[100];
```

malloc()使程序能够根据需要分配存储空间。当然，可用的存储空间是有限的，它取决于计算机上安装的内存量以及其他程序所需的内存空间。如果没有足够的内存可用，malloc()将返回 null（0）。程序应检查 malloc()返回的值，以便知道是否成功地分配了所需的内存。您应该总是检查 malloc()返回的值是否等于符号常量 NULL，该常量是在头文件 stdlib.h 中定义的。程序清单 10.3 演示了 malloc()的用法。任何使用了 malloc()的程序都必须包含头文件 stdlib.h。



提示：在前面的范例中，为字符分配内存时，使用的是字面值。所需分配的内存量为要为之分配内存的数据类型的长度与数目的乘积。前面分配内存时，假设一个字符占用一个字节。如果字符占用多个字节，则上述范例将重写内存中的其他区域。例如：

```
ptr = malloc(100);
```

实际上应这样编写：

```
ptr = malloc( 100 * sizeof(char));
```

程序清单 10.3　　memalloc.c：使用 malloc()函数为字符串数据分配内存空间

```
1:  /* Demonstrates the use of malloc() to allocate storage */
2:  /* space for string data. */
3:
4:  #include <stdio.h>
5:  #include <stdlib.h>
6:
7:  char count, *ptr, *p;
8:
9:  int main( void )
10: {
11:    /* Allocate a block of 35 bytes. Test for success. */
12:    /* The exit() library function terminates the program. */
13:
14:    ptr = malloc(35 * sizeof(char));
15:
16:    if (ptr == NULL)
17:    {
18:       puts("Memory allocation error.");
19:       return 1;
20:    }
21:
22:    /* Fill the string with values 65 through 90, */
23:    /* which are the ASCII codes for A-Z. */
24:
25:    /* p is a pointer used to step through the string. */
26:    /* You want ptr to remain pointed at the start */
27:    /* of the string. */
28:
29:    p = ptr;
30:
31:    for (count = 65; count < 91 ; count++)
32:       *p++ = count;
33:
34:    /* Add the terminating null character. */
35:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
36:    *p = '\0';
37:
38:    /* Display the string on the screen. */
39:
40:    puts(ptr);
41:
42:    free(ptr);
43:
44:    return 0;
45: }
```

该程序的输出如下：

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

分析：该程序以简单的方式使用了 malloc()。虽然程序看起来很长，但其中包含很多注释。第 1、2、11、12、22～27、34 和 38 行都是注释，它们详细地介绍程序完成的每一项工作。第 5 行包含了头文件 stdlib.h，这是 malloc()所要求的；而第 4 行包含了头文件 stdio.h，这是 puts()函数所要求的。第 7 行声明了两个指针和一个字符变量，供程序后面使用。这些变量都没有被初始化，因此还不能使用它们。

第 14 行调用 malloc()函数，并将参数 35 * sizeof(char)传递给它。可以传递参数 35 吗？可以，但这样将假定在所有运行该程序的计算机中，字符变量都只占用一个字节。第 3 天的课程介绍过，变量的长度可能随编译器而异。使用运算符 sizeof 可以确保代码的可移植性。

绝不要认为 malloc()能成功地分配所需的内存。第 16 行演示了如何检查 malloc()是否成功地分配了所需的内存。如果是，则 ptr 将指向分配的内存；否则将 ptr 将为空，在这种情况下，将执行第 18 和 19 行——显示一条错误消息，并妥善地结束程序。

第 29 行初始化第 7 行声明的另一个指针 p，将 ptr 的值赋给 p。一个 for 循环使用这个新的指针将值存储到分配的内存中。在第 31 行，count 被初始化为 65，然后被不断递增，直到等于 91。每次循环时，将 count 的值赋给 p 指向的地址。这意味着每个值将被依次放到内存中。

您应该注意到了，赋给 count 的是数字，而 count 是一个 char 变量。还记得前面有关 ASCII 字符及其对应数值的讨论吗？数字 65 对应于 A，66 对应于 B，67 对应于 C，依此类推。将所有的字母都存储到指针指向的内存块中后，for 循环结束。第 36 行在 p 指向的地址加上一个空字符，这样便可以将这些字符作为一个字符串使用了。ptr 仍然指向第一个字符（A），因此将它作为一个字符串使用，可以打印所有的字符，直到遇到空字符为止。第 40 行使用 puts()来证明了这一点。

第 42 行是一个新函数 free()。当您动态分配内存时，使用完毕后，应释放它们。free()函数将分配的内存归还给操作系统。因此第 42 行将分配并赋给 ptr 的内存还给系统。

应　该	不 应 该
	分配的内存量不应超过所需。并非每台计算机都有大量的内存，因此应节俭使用。 不要将过大的字符串赋给数组。例如，在下述声明中： `char a_string[] = "NO";` a_string 指向 NO。如果试图将“YES”赋给该数组，将可能导致严重的问题。该数组最初只能存储 3 个字符——N、O 和空字符，而字符串“YES”有 4 个字符——Y、E、S 和空字符。这样第 4 个字符将覆盖后面的内容，而您对此却一无所知。

10.6　显示字符串和字符

如果程序使用了字符串数据，则可能需要将这些数据显示在屏幕上。显示字符串的工作通常是使用函数 puts()和 printf()来完成的。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

10.6.1　puts()函数

本书前面的一些程序已经使用过库函数 puts()。该函数将一个字符串放置到屏幕上，因此名为 puts。puts()只接受一个参数——指向要显示的字符串的指针。由于字面字符串是一个指向字符串的指针，因此 puts()可用于显示字面字符串和字符串变量。puts()显示完字符串后，自动换行，因此使用 puts()显示的每个字符串都单独占一行。

程序清单 10.4 演示了 puts()函数的用法。

程序清单 10.4　　puts.c：使用 puts()函数将文本显示到屏幕上

```
1: /* Demonstrates displaying strings with puts(). */
2:
3: #include <stdio.h>
4:
5: char *message1 = "C";
6: char *message2 = "is the";
7: char *message3 = "best";
8: char *message4 = "programming";
9: char *message5 = "language!!";
10:
11: int main( void )
12: {
13:    puts(message1);
14:    puts(message2);
15:    puts(message3);
16:    puts(message4);
17:    puts(message5);
18:
19:    return 0;
20: }
```

该程序的输出如下：

```
C
is the
best
programming
language!!
```

分析：该程序很容易理解。由于 puts()是一个标准的输出函数，因此需要包含头文件 stdio.h（第 3 行）。第 5～9 行声明并初始化了 5 个不同的变量，其中每个变量都是字符指针（字符串变量）。第 13～17 行使用 puts()函数来打印各个字符串。

10.6.2　printf()函数

也可以使用库函数 printf()来显示字符串。第 7 天的课程介绍过，printf()使用一个格式化字符串和转换说明符来格式化其输出。要使用字符串，需要使用转换说明符%s。

当 printf()遇到其格式化字符串中的%s，它将%s 与参数列表中相应的参数进行匹配。对于字符串，参数必须是指向要显示的字符串的指针。printf()函数将字符串显示到屏幕上，直到遇到该字符串的结束空字符。下面是一个例子：

```
char *str = "A message to display";
printf("%s", str);
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

您也可以显示多个字符，并同时显示字面文本和/或数值变量：

```
char *bank = "First Federal";
char *name - "John Doe";
int balance = 1000;
printf("The balance at %s for %s is %d.", bank, name, balance);
```

上述代码的输出为：

```
The balance at First Federal for John Doe is 1000.
```

就现在而言，这些信息对于您在程序中显示字符串数据足够了。有关如何使用 printf()的详细信息，请参阅第 14 天的课程。

10.7 从键盘读取字符串

除了显示字符串外，程序还经常需要从键盘读取用户输入的字符串数据。C 语言库中有两个可用来完成这项工作的函数：gets()和 scanf()。然而，从键盘读取字符串之前，必须有用于存储它们的地方。可以使用前面介绍的两种方法之一（数组声明或 malloc()函数）为字符串提供存储空间。

10.7.1 使用 gets()函数输入字符串

函数 gets()从键盘读取一个字符串。当 gets()函数被调用时，它不断从键盘读取字符，直到遇到换行符（通过按 Enter 键生成）为止。该函数丢弃换行符，添加一个空字符，然后将字符串返回给调用程序。字符串被存储到传递给 gets()的 char 指针指向的位置。使用了 gets()函数的程序必须包含头文件 stdio.h。程序清单 10.5 是一个使用 gets()函数的范例。

程序清单 10.5　　gets.c：使用 gets()从键盘输入字符串数据

```
1:  /* Demonstrates using the gets() library function. */
2:
3:  #include <stdio.h>
4:
5:  /* Allocate a character array to hold input. */
6:
7:  char input[81];
8:
9:  int main( void )
10: {
11:     puts("Enter some text, then press Enter: ");
12:     gets(input);
13:     printf("You entered: %s\n", input);
14:
15:     return 0;
16: }
```

该程序的运行情况如下：

```
Enter some text, then press Enter:
This is a test
You entered: This is a test
```

分析：在该程序中，传递给 gets()的参数为 input——一个 char 数组的名称，因此是指向该数组的第一个元素的指针。第 7 行将该数组声明为包含 81 个元素。由于在大多数的计算机屏幕上，一行最多包含 80 个字符，因此这样的数组长度足以存储一整行的输入（加上 gets()在末尾添加的空字符）。

函数 gets()有一个返回值，但该程序没有使用它。gets()函数返回一个 char 指针，该指针指向输入的字符

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

串的存储地址。这与传递给 gets()的值相同，但提供这样的返回值，让程序能够检查输入的是否为空行，程序清单 10.6 演示了这一点。

程序清单 10.6　　getback.c：使用 gets()函数的返回值检查用户输入的是空行

```
1:  /* Demonstrates using the gets() return value. */
2:
3:  #include <stdio.h>
4:
5:  /* Declare a character array to hold input, and a pointer. */
6:
7:  char input[257], *ptr;
8:
9:  int main( void )
10: {
11:     /* Display instructions. */
12:
13:     puts("Enter text a line at a time, then press Enter.");
14:     puts("Enter a blank line when done.");
15:
16:     /* Loop as long as input is not a blank line. */
17:
18:     while ( *(ptr = gets(input)) != NULL)
19:        printf("You entered %s\n", input);
20:
21:     puts("Thank you and good-bye\n");
22:
23:     return 0;
24: }
```

该程序的运行情况如下：

```
Enter text a line at a time, then press Enter.
Enter a blank line when done.
First string
You entered First string
Two
You entered Two
Bradley L. Jones
You entered Bradley L. Jones

Thank you and good-bye
```

分析：从上述输出可以知道程序是如何运行的。如果用户输入一个空行（即只是按下 Enter 键）来响应第 18 行，字符串仍将被存储（不包含任何字符），并在末尾加上空字符。由于字符串的长度为 0，因此空字符被存储在开头的位置。函数 gets()的返回值指向的就是这个位置，因此检测这个位置时，将发现一个空字符，这样您便知道用户输入的是一个空行。

在程序清单 10.6 中，这种测试是在 while 语句（第 18 行）中进行的。这个语句有点复杂，请详细查看其细节。图 10.1 说明了其中的各个组成部分。



警告：由于并非总能知道 gets()将读取多少个字符，而 gets()将不断地存储字符，这可能超出缓冲区的末尾，因此使用该函数时一定要小心。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

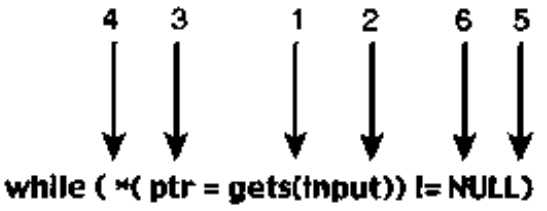


图 10.1　检查输入是否为空行的 while 语句的组成部分

1．gets()函数不断从键盘读取输入，直到遇到换行符；

2．将输入的字符串（删除换行符，加上空字符）存储到 input 指向的内存单元；

3．将字符串的地址（input 的值）返回给指针 ptr；

4．赋值语句是一个表达式，值为赋值运算符左边的变量的值。因此整个表达式 ptr = gets(input)的值为 ptr 的值。用花括号将该表达式括起，并使用间接运算符来处理它，可以获得存储在 ptr 指向的地址处的值。这是输入的字符串中的第一个字符。

5．NULL 是头文件 stdio.h 中定义的一个符号常量，其值为空字符（0）；

6．如果输入的字符串的第一个字符不是空字符（即输入的不是空行），则比较的结果为 true，while 循环将执行；否则比较的结果为 false，while 循环将结束。

使用 gets()或其他使用指针来存储数据的函数时，指针一定要指向已分配的空间。人们很容易犯下面这样的错误：

```
char *ptr;
gets(ptr);
```

指针 ptr 被声明了，但未被初始化。它指向某个地方，但您不知道是什么地方。gets()函数不知道 ptr 未被初始化为指向某个地方，因此它将输入的字符串存储在 ptr 指向的地址中。这样，该字符串将可能覆盖重要的信息，如程序代码或操作系统代码。大多数编译器不会捕获这样的错误，因此您必须小心。

gets()函数的语法如下：

```
#include <stdio.h>
char *gets(char *str);
```

函数 gets()从标准输入设备（通常是键盘）读取一个字符串。该字符串由换行符之前的所有字符组成，末尾被加上一个空字符。

然后，gets()函数返回一个指针，它指向前面读取的字符串。如果读取字符串时出错，gets()将返回 null。

下面是一个使用 gets()函数的范例：

```
/* gets() example */
#include <stdio.h>
char line[256];
void main( void )
{
      printf( "Enter a string:\n");
      gets( line );
      printf( "\nYou entered the following string:\n" );
      printf( "%s\n", line );
}
```

10.7.2　使用 scanf()函数输入字符串

第 7 天的课程介绍过，库函数能够从键盘读取数值数据；该函数也可用来输入字符串。scanf()使用一个格式化字符串，该字符串告诉它如何读取输入的信息。要读取字符串，应在 scanf()的格式化字符串中包含说明符%s。与 gets()一样，也需要将指向字符串存储位置的指针传递给 scanf()。

scanf()如何确定字符串的开始和结束位置呢？开始位置是遇到的第一个非空白字符；结束位置可通过两种方式来指定。如果在格式化字符串中使用的是%s，则字符串到下一个空白字符（空格、制表符或换行符）之前结束；如果使用的是%ns（其中 n 为一个整型常量，表示字段的宽度）则 scanf()读取 n 个字符或直到遇

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

到下一个空白字符。

可以在格式化字符串中包含多个%s 来读取多个字符串。对于格式化字符串中的每个%s，sacnf()都按前面介绍的规则来读取输入中的字符串。例如对于下面的语句：

```
scanf("%s%s%s", s1, s2, s3);
```

如果用户输入 January February March 来做出响应，则 January 将被赋给字符串 s1，February 将被赋给 s2，March 将被赋给 s3。

使用字段宽度说明符的情况如何呢？对于下面的语句：

```
scanf("%3s%3s%3s", s1, s2, s3);
```

如果用户输入 September 来响应，则 Sep 被赋给 s1，tem 被赋给 s2，ber 被赋给 s3。

如果用户输入的字符串比 scanf()函数期望的少，情况将如何呢？在这种情况下，scanf()等待用户输入字符串，程序将暂停，直到用户输入所需的字符串为止。例如，对于下面的语句：

```
scanf("%s%s%s", s1, s2, s3);
```

如果用户输入 January February 来响应，则程序将暂停，等待用户输入 scanf()的格式化字符串中指定的第三个字符串。如果用户输入的字符串数比要求的多，则多出的字符串将留在输入缓存中，接下来的 scanf()或其他输入语句将读取这些字符串。例如，对于下面的语句：

```
scanf("%s%s", s1, s2);
scanf("%s", s3);
```

如果用户输入 January February March 来响应，则第一个 scanf()调用将 January 赋给 s1，February 赋给 s2；第二个 scanf()调用接着将 March 赋给 s3。

scanf()有一个返回值——成功输入的字符串数目，通常将其忽略。只读取文本时，gets()函数优于 scanf()；而同时读取文本和数值数据时，最好使用 scanf()。程序清单 10.7 说明了这一点。第 7 天的课程介绍过，使用 scanf()将输入的数据赋给数值变量时，必须使用地址运算符（&）。

程序清单 10.7　　inputs.c：使用 scanf()输入数值和文本数据

```
1:  /* Demonstrates using scanf() to input numeric and text data. */
2:
3:  #include <stdio.h>
4:
5:  char lname[257], fname[257];
6:  int count, id_num;
7:
8:  int main( void )
9:  {
10:     /* Prompt the user. */
11:
12:     puts("Enter last name, first name, ID number separated");
13:     puts("by spaces, then press Enter.");
14:
15:     /* Input the three data items. */
16:
17:     count = scanf("%s%s%d", lname, fname, &id_num);
18:
19:     /* Display the data. */
20:
21:     printf("%d items entered: %s %s %d \n", count, fname, lname, id_num);
22:
23:     return 0;
24: }
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

该程序的运行情况如下：

```
Enter last name, first name, ID number separated
by spaces, then press Enter.
Jones Bradley 12345
3 items entered: Bradley Jones 12345
```

分析：scanf()使用变量的地址作为参数。在程序清单 10.7 中，lname 和 fname 是指针（即地址），因此无需对它们使用地址运算符；而 id_num 是一个常规变量，因此第 17 行将其传递给 scanf()时，使用了地址运算符。

有些程序员认为，使用 scanf()输入数据容易出错；因此它们喜欢使用 gets()来输入所有的数据（数值数据和字符串），然后让程序将数字分离出来，并将其转换为数值。这种技术超出了本书的范围，但可作为一个不错的编程练习。为完成这种任务，必须使用第 17 天课程中将介绍的字符串操纵函数。

10.8 总 结

今天的课程介绍了 char 数据类型。char 变量的用途之一是用于存储单个字符。字符通常被存储为数字：分配给每个字符的 ASCII 码，因此也可以使用 char 变量来存储小型整数。有两种 char 数据类型：signed char 和 unsigned char。

字符串是一个以空字符结尾的字符序列，可用于存储文本数据。C 语言将字符串存储为 char 数组。要存储一个长度为 n 的字符串，需要使用一个包含 n+1 个元素的 char 数组。

可以使用诸如 malloc()等内存分配函数来提高程序的动态性。通过使用 malloc()，可以为程序分配合适的内存量；如果没有这样的函数，您将需要猜测程序所需的内存量。您的估计可能过高，因此分配过多的内存。使用完分配的内存后，应使用 free()函数将其返回给系统。

10.9 问与答

问：字符串与字符数组之间有何区别？

答：字符串是一个以空字符结尾的字符序列；而字符数组是一个字符序列。因此，字符串是一个以空字符结尾的字符数组。

当您定义一个 char 数组时，实际为该数组分配的存储空间为指定的数组长度，而不是该长度减 1。您在该数组中存储的字符串的长度不能超过数组长度。下面是一个例子：

```
char state[10]= "Minneapolis";  /* Wrong! String longer than array. */
char state2[10]= "MN";          /* OK, but wastes space because */
                                /* string is shorter than array. */
```

但如果定义一个 char 指针，则不存在这样的限制。在这种情况下，指针变量只是用于存储指针的存储空间，实际的字符串将被存储在内存的其他地方（但您无需关心它到底被存储在什么地方）。使用指针时，不存在长度限制或浪费空间的情况，实际的字符串被存储在其他地方。指针可以指向任意长度的字符串。

问：为何不声明一个非常大的数组来存储值，而要使用诸如 malloc()等内存分配函数？

答：虽然声明一个大型数组更容易，但这种使用内存的方式效率不高。编写小型程序（如今天课程中的程序）时，使用诸如 malloc()等函数（而不是数组）的意义不大；但当程序非常大时，应根据需要来分配内存。使用完内存后，可以释放它，从而将它归还给系统。将内存释放后，程序其他部分中的某些变量或数组可以使用它（有关如何释放内存，请参阅第 20 天的课程）。

问：是否所有的计算机都支持扩展 ASCII 字符集？

答。不是。大多数 PC 支持扩展 ASCII 字符集。有些老式 PC 不支持，但这种老式 PC 已越来越少。大多数程序员使用扩展字符集中的线字符和块字符。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

另外，很多国际字符集包含了 ASCII 中没有的字符。对于这种字符，通常使用 wchar_t 变量（而不是 char 变量）来存储。wchar_t 类型是在头文件 stddef.h 中定义的，可用来存储大型字符集中的字符。有关如何使用 wchar_t 和其他字符集的详细信息，请参阅 ANSI 文档。

问：将一个长度大于数组的字符串存储到数组中，情况将如何？

答：可能导致难以查出的错误。在 C 语言中，可以这样做，但将把字符数组后面的内存中的数据覆盖掉。这些内存可能没有使用，也可能包含其他数据或一些至关重要的系统信息。结果将取决于覆盖的内容是什么。通常暂时不会出现问题，但不要这样做。

10.10　作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

10.10.1　小测验

1. ASCII 字符集的数值范围是什么？
2. C 编译器遇到用单引号括起的字符时，如何解释它？
3. C 语言是如何定义字符串的？
4. 字面字符串是什么？
5. 要存储一个包含 n 个字符的字符串，需要一个包含 n+1 个元素的字符数组。为何需要多出一个元素？
6. C 编译器遇到字面字符串时，如何解释它？
7. 请根据附录 A 中的 ASCII 表，指出下面字符对应的数值：
 a. a
 b. A
 c. 9
 d. 空格
 e. ╬
 f. ♠
8. 请根据附录 A 中的 ASCII 表，指出下面的数值对应的字符：
 a. 73
 b. 32
 c. 99
 d. 97
 e. 110
 f. 0
 g. 2
9. 对于下面的各个变量，将分配多少字节的存储空间？（假设一个字符占一字节）
 a. `char *str1 = { "String 1" };`
 b. `char str2[] = { "String 2" };`
 c. `char string3;`
 d. `char str4[20] = { "This is String 4" };`
 e. `char str5[20];`
10. 给定声明 **char *string = "A string!";**，下述各个表达式对应的值是什么？
 a. `string[0]`
 b. `*string`
 c. `string[9]`

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

d. `string[33]`

e. `*string+8`

f. `string`

10.10.2 练习

1. 编写一行代码，声明一个名为 letter 的 char 变量，并将其初始化为字符$。

2. 编写一行代码，声明一个 char 数组，并将其初始化为“Pointers are fun!”，该数组的大小为刚好能够存储上述字符串。

3. 编写一行代码，为字符串“Pointers are fun!”分配存储空间，但不使用数组。

4. 编写这样的代码：为一个包含 80 个字符的字符串分配空间，然后从键盘输入一个字符串，并将其存储到分配的存储空间中。

5. 编写一个函数，将一个字符数组复制到另一个字符数组中（提示：参考第 9 天课程中的程序）。

6. 编写一个函数，它接受两个字符串参数，并返回一个指针，该指针指向较长的那个字符串。

7. 选做题：编写一个函数，它接受两个字符串参数，将它们连接成一个字符串，并使用函数 malloc() 为该字符串分配足够的存储空间，然后返回一个指向该字符串的指针。

例如，如果传递的是“Hello”和“World!”，该函数将返回一个指向“Hello World!”的指针。使合并和的字符串作为第三个字符串最为简单（可以使用练习 5 和练习 6 中编写的代码）。

8. 排错：下面代码有错误吗？

```
char a_string[10] = "This is a string";
```

9. 排错：下面代码有错误吗？

```
char *quote[100] = { "Smile, Friday is almost here!" };
```

10. 排错：下面代码有错误吗？

```
char *string1;
char *string2 = "Second";
string1 = string2;
```

11. 排错：下面代码有错误吗？

```
char string1[];
char string2[] = "Second";
string1 = string2;
```

12. 选做题：根据 ASCII 表编写一个程序，使用双线字符（double-line）在屏幕上打印一个方框。



警告：今天的课程使用函数 malloc()来动态地分配内存。动态地分配内存时，使用完内存后，应将其释放，以归还给计算机系统。可以使用 free()函数来释放动态分配的内存，这将在第 20 天的课程中介绍。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第 11 天课程　结构、共用体和 TypeDef

通过使用结构，可以简化很多编程任务。结构是程序员根据编程需求设计的一种数据存储类型。今天介绍以下内容：

- 何为简单结构和复杂结构？
- 如何定义和声明结构？
- 如何存取结构中的数据？
- 如何创建包含数组和结构数组的结构？
- 如何声明结构中的指针和指向结构的指针？
- 如何定义、声明和使用共用体？
- 如何对结构使用类型定义？

11.1　简单结构

结构是为易于操作而被组合在一起的一个或多个变量。不同于数组，结构中的变量的数据类型可以不同。结构可以包含任何数据类型的变量，包括数组和其他结构。结构中的变量被称为结构的成员。下一节将介绍一个简单的结构。

您应首先学习简单结构。C 语言中没有简单结构和复杂结构之分，但这样区分，解释结构起来将更容易。

11.1.1　定义和声明结构

编写图形程序时，需要处理屏幕上的点的坐标。屏幕坐标用 x 和 y 值表示，前者为水平位置，后者为垂直位置。可以定义一个包含屏幕位置的 x 和 y 值的 coord 结构，如下所示：

```
struct coord
{
    int x;
    int y;
};
```

关键字 struct 用于标识结构定义的开始，后面跟结构的名称。这种规则与其他数据类型相同。结构的名称也叫结构的标记（tag）或类型名。后面将介绍如何使用标记。

结构标记的后面是左花括号。花括号内是结构的成员变量列表，对于其中的每个成员，都必须指定其数据类型和名称。

前面的代码定义了一种名为 coord 的结构类型，它包含两个整型变量：x 和 y，上述对结构 coord 及其成员（x 和 y）的声明并不会创建结构 coord 或变量 x 和 y 的实例，换句话说，它没有声明任何结构（为其预留存储空间）。声明结构的方式有两种。一种是在结构定义的后面加上一个或多个变量名，如下所示：

```
struct coord {
```

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
    int x;
    int y;
} first, second;
```

上述语句定义了结构类型coord，并声明了两个这样的结构：first和second。first和second都是coord类型的实例，它们都包含两个整型成员：x和y。

这种方法在定义结构的同时声明结构。另一种方法是将定义和声明分开。下面的语句也声明了两个coord类型的实例：

```
struct coord {
    int x;
    int y;
};
/* Additional code may go here */
struct coord first, second;
```

在这个例子中，coord结构的定义和变量声明被分开。单独声明变量时，使用关键字struct，然后是结构名和变量名。

11.1.2 存取结构的成员

对于结构中的每个成员，可以像同类型的其他变量那样使用它们。结构成员是通过在结构名和成员名之间使用结构成员运算符（也叫句点运算符）来存取的。因此，要让结构first指的是屏幕上坐标为50和100的点，可以这样编写代码：

```
first.x = 50;
first.y = 100;
```

要显示存储在结构second中的屏幕位置，可以这样编写代码：

```
printf("%d,%d", second.x, second.y);
```

您可能会问，与使用变量相比，使用结构有何优点？一个主要的优点是，可以使用简单的赋值语句在同类型的结构之间复制信息。继续以前面的例子为例，语句：

```
first = second;
```

与下面的语句等价：

```
first.x = second.x;
first.y = second.y;
```

当程序使用了包含很多成员的复杂结构时，这种表示法可以节省大量的时间。当您学习了一些高级技术后，结构的其他优势将显现出来。通常，当需要把不同类型的信息作为一个整体进行处理时，结构将很有用。例如，在邮寄名单数据库中，每个条目是一个结构，每项信息（姓名、地址、城市等）是一个结构成员。

程序清单11.1演示了以上介绍的有关结构的知识。它虽然不太实用，但说明了简单结构的要点。

程序清单11.1　　simple.c：声明并使用简单结构

```
1:  /* simple.c - Demonstrates the use of a simple structures*/
2:
3:  #include <stdio.h>
4:
5:  int length, width;
6:  long area;
7:
8:  struct coord{
9:      int x;
10     int y;
11: } myPoint;
12:
13: int main( void )
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
14: {
15:     /* set values into the coordinates */
16:     myPoint.x = 12;
17:     myPoint.y = 14;
18:
19:     printf("\nThe coordinates are: (%d, %d).",
20:           myPoint.x, myPoint.y);
21:
22:     return 0;
23: }
```

该程序的输出如下：

```
The coordinates are: (12, 14).
```

分析：该程序定义了一个简单的结构，用于存储点的坐标。该结构与前面介绍的相同。第 8 行使用了关键字 struct，后面跟结构名 coord。接下来第 9-11 行定义了结构体。该结构包含两个成员：x 和 y，它们的数据类型都是 int。

第 11 行定义了一个名为 myPoint 的 coord 结构的实例。也可以在单独一行中完成这种声明，如下所示：

```
struct coord myPoint;
```

第 16 和 17 行给 myPoint 的成员赋值。正如前面指出的，给成员赋值时，使用的是结构变量名和成员名，并在它们之间加上成员运算符（.）。第 19 和 20 行在 printf()函数中使用了这些成员。

定义结构的语法如下：

```
struct tag {
    structure_member(s);
    /* additional statements may go here */
} instance;
```

关键字 struct 用于定义结构。结构是为易于操作而被组合在一起的一个或多个变量，其中变量的数据类型不要求相同，也不要求是简单变量。结构中可以包含数组、指针和其他结构。

关键字 struct 标识结构定义的开始，后面是给结构指定的名称。结构名的后面是结构成员，用花括号括起。接下来还可以定义结构的声明（即实例）。定义结构时，如果没有同时声明其实例，则它只是一个模板（或定义），后面可以使用它来声明结构。模板的格式如下：

```
struct tag {
    structure_member(s);
    /* additional statements may go here */
};
```

使用模板的格式如下：

```
struct tag instance;
```

要使用这种方式，必须以前已定义了结构，并给它指定了名称。

下面是三个定义结构的范例：

范例 1：

```
/* Declare a structure template called SSN */
struct SSN {
    int first_three;
    char dash1;
    int second_two;
    char dash2;
    int last_four;
}
/* Use the structure template */
struct SSN customer_ssn;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

范例2：

```
/* Declare a structure and instance together */
struct date {
    char month[2];
    char day[2];
    char year[4];
} current_date;
```

范例3：

```
/* Declare and initialize a structure */
struct time {
    int hours;
    int minutes;
    int seconds;
} time_of_birth = { 8, 45, 0 };
```

11.2 复杂结构

学习简单结构后，接着学习更有趣、更复杂的结构类型。这些结构将其他结构和数组作为其成员。

11.2.1 包含其他结构的结构

如前面所介绍，C 语言结构可以包含任何数据类型。例如，一个结构可以包含其他结构。将前一个例子进行扩展来说明这一点。

假设图形程序需要处理矩形。可以使用对角顶点的坐标来定义矩形。前面已经介绍了如何定义一个用于存储点的坐标的结构。要定义矩形，需要两个这样的结构。您可以定义一个这样的结构（假设已经定义了 coord 结构）：

```
struct rectangle {
    struct coord topleft;
    struct coord bottomrt;
};
```

该语句定义了一个 rectangle 结构，它包含两个 coord 结构：topleft 和 bottomrt。

上述语句只是定义了结构类型 rectangle。要声明一个这样的结构，必须编写一条下面这样的语句：

```
struct rectangle mybox;
```

可以将定义和声明合并到一起，就像 coord 那样：

```
struct rectangle {
    struct coord topleft;
    struct coord bottomrt;
} mybox;
```

要存取实际的位置（int 成员），必须使用两个成员运算符（.）。因此，下面的表达式：

```
mybox.topleft.x
```

指的是 rectangle 结构 mybox 的成员 topleft 的成员 x。要使用两对坐标来定义一个矩形，可以这样编写代码：

```
mybox.topleft.x = 0;
mybox.topleft.y = 10;
mybox.bottomrt.x = 100;
mybox.bottomrt.y = 200;
```

上述代码也许有些令人迷惑，参考图 11.1 将有助于理解它们。该图说明了 rectangle 结构及其包含的两个 coord 结构与 coord 结构包含的 int 变量之间的关系。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

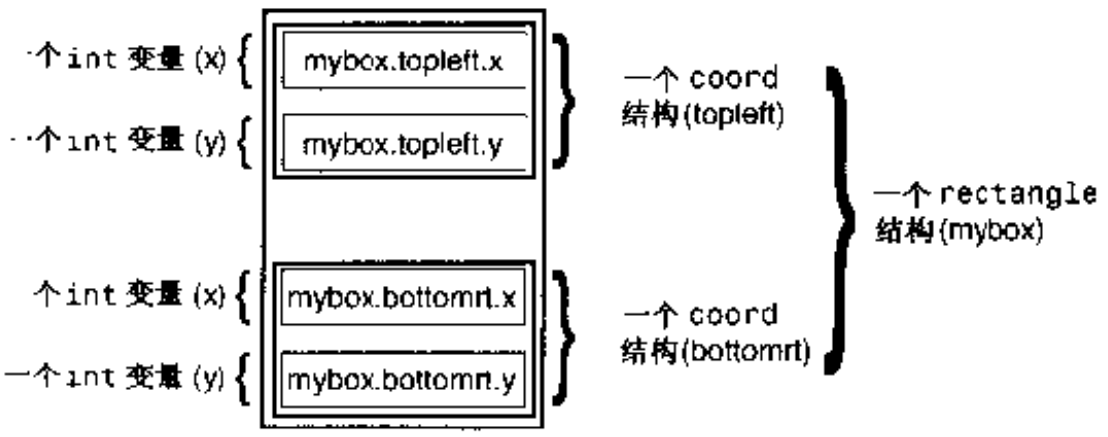


图 11.1　结构、其包含的结构以及被包含的结构的成员之间的关系

程序清单 11.2 演示了如何使用包含其他结构的结构。该程序清单让用户输入矩形的对角顶点的坐标，然后计算并显示该矩形的面积。程序开头的注释（第 3~8 行）说明了该程序做出的假设。

程序清单 11.2　　struct.c：包含其他结构的结构

```
1:  /* Demonstrates structures that contain other structures. */
2:
3:  /* Receives input for corner coordinates of a rectangle and
4:     calculates the area. Assumes that the y coordinate of the
5:     lower-right corner is greater than the y coordinate of the
6:     upper-left corner, that the x coordinate of the lower-
7:     right corner is greater than the x coordinate of the upper-
8:     left corner, and that all coordinates are positive. */
9:
10: #include <stdic.h>
11:
12: int length, width;
13: long area;
14:
15: struct coord{
16:     int x;
17:     int y;
18: };
19:
20: struct rectangle{
21:     struct coord topleft;
22:     struct coord bottomrt;
23: } mybox;
24:
25: int main( void )
26: {
27:     /* Input the coordinates */
28:
29:     printf("\nEnter the top left x coordinate: ");
30:     scanf("%d", &mybox.topleft.x);
31:
32:     printf("\nEnter the top left y coordinate: ");
33:     scanf("%d", &mybox.topleft.y);
34:
35:     printf("\nEnter the bottom right x coordinate: ");
36:     scanf("%d", &mybox.bottomrt.x);
37:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
38:     printf("\nEnter the bottom right y coordinate: ");
39:     scanf("%d", &mybox.bottomrt.y);
40:
41:     /* Calculate the length and width */
42:
43:     width = mybox.bottomrt.x - mybox.topleft.x;
44:     length = mybox.bottomrt.y - mybox.topleft.y;
45:
46:     /* Calculate and display the area */
47:
48:     area = width * length;
49:     printf("\nThe area is %ld units.\n", area);
50:
51:     return 0;
52: }
```

该程序的运行情况如下：

```
Enter the top left x coordinate: 1

Enter the top left y coordinate: 1

Enter the bottom right x coordinate: 10

Enter the bottom right y coordinate: 10

The area is 81 units.
```

分析：第15～18行定义了coord结构，它包含两个成员x和y；第20～23行定义了rectangle结构，并声明了该结构的一个实例mybox。rectangle结构包含两个成员：topleft和bottomrt，它们都是coord结构。

第29～39行读取mybox结构的值。首先，您可能认为mybox只有两个值需要读取，因为它只有两个成员。然而，mybox的每个成员本身又有成员。topleft和bottomrt都有两个成员：x和y。因此，需要读取4个值。读取成员的值后，程序使用结构名和成员名来计算矩形的面积。使用x和y的值时，必须提供结构实例的名称。由于x和y都属于一个结构，而该结构又属于另一个结构，因此计算时必须使用这两个结构实例的名称：mybox.bottomrt.x、mybox.bottomrt.y、mybox.topleft.x和mybox.topleft.y。

C语言没有限制结构的嵌套层数，但ANSI标准最多只支持63层。只要内存足够多，您可以定义嵌套多层的结构。当然，这有一定的限度，超过这一限度将没有好处。在任何C语言程序中，嵌套很少超过3层。

11.2.2 包含数组的结构

结构可以将一个或多个数组作为其成员。其中的数组可以是任何数据类型（int、char等）。例如，下面的语句：

```
struct data
{
    int  x[4];
    char y[10];
};
```

定义了一个data结构，该结构包含一个名为x的整型数组和一个名为y的字符数组，其中前者包含4个元素，后者包含10个元素。然后，便可以声明一个名为record的data结构，如下所示：

```
struct data record;
```

该结构的组织方式如图11.2所示。在该图中，数组x的元素占用的空间为数组y的元素的两倍，这是因为int类型通常需要2字节的存储空间，而char类型只需要一个字节（这在第3天的课程中介绍过）。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

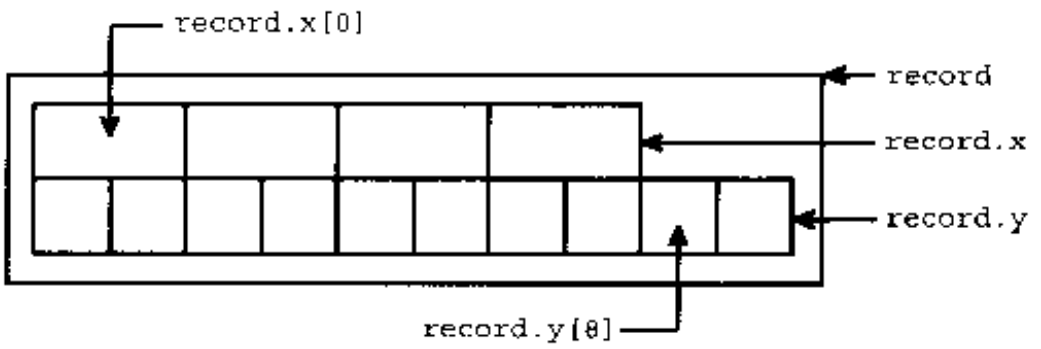


图 11.2　包含两个数组的结构的组织方式

要存取结构中的数组元素，可以使用成员运算符和数组下标：

```
record.x[2] = 100;
record.y[1] = 'x';
```

您可能还记得，字符数组最常用于存储字符串，另外，不带方括号的数组名是指向该数组的指针。对于作为结构成员的数组，情况也是如此。因此，下面的表达式：

```
record.y
```

是一个指针，它指向结构 record 中的数组 y[]的第一个元素。因此，可以使用下面的语句将数组 y 的内容打印到屏幕上：

```
puts(record.y);
```

请看另一个例子。程序清单 11.3 使用了一个结构，该结构包含一个 float 变量和两个 char 数组。

程序清单 11.3　　**array.c：将数组作为成员的结构**

```
1:  /* Demonstrates a structure that has array members. */
2:
3:  #include <stdio.h>
4:
5:  /* Define and declare a structure to hold the data. */
6:  /* It contains one float variable and two char arrays. */
7:
8:  struct data{
9:      float amount;
10:     char fname[30];
11:     char lname[30];
12: } rec;
13:
14: int main( void )
15: {
16:     /* Input the data from the keyboard. */
17:
18:     printf("Enter the donor's first and last names,\n")
19:     printf("separated by a space: ");
20:     scanf("%s %s", rec.fname, rec.lname);
21:
22:     printf("\nEnter the donation amount: ");
23:     scanf("%f", &rec.amount);
24:
25:     /* Display the information. */
26:     /* Note: %.2f specifies a floating point value */
27:     /* to be displayed with two digits to the right */
28:     /* of the decimal point. */
29:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn


```
30:     /* Display the data on the screen. */
31:
32:     printf("\nDonor %s %s gave $%.2f.\n", rec.fname, rec.lname,
33:            rec.amount);
34:
35:     return 0;
36: }
```

该程序的运行情况如下：

```
Enter the donor's first and last names,
separated by a space: Bradley Jones

Enter the donation amount: 1000.00

Donor Bradley Jones gave $1000.00.
```

分析：该程序使用了一个结构，该结构包含两个数组成员：fname[30]和 lname[30]，它们都是字符数组，分别用于存储姓和名。第 8～12 行定义了一个 data 结构，它包含两个字符数组（fname 和 lname）和一个名为 amount 的 float 变量。该结构适合用来存储人的姓名和一个值（如给慈善机构的捐款金额）。

第 12 行声明了一个名为 rec 的 data 结构的实例。程序的其他代码使用 rec 来让用户输入值（第 18～23 行），然后打印这些值（第 32～33 行）。

11.3 结构数组

既然结构可以包含数组，数组是否可以包含结构呢？可以，实际上结构数组是一个非常强大的编程工具。

前面介绍了如何根据程序要使用的数据来定义结构。通常，程序需要使用多个数据实例。例如，在维护电话号码簿的程序中，您可以定义一个结构来存储每个人的姓名和电话号码：

```
struct entry
{
    char fname[10];
    char lname[12];
    char phone[8];
};
```

电话号码簿包含多个条目，因此单个 entry 结构的实例用处不大。您需要一个由 entry 结构组成的数组。定义结构后，便可以定义一个由该结构组成的数组：

```
struct entry list[1000];
```

上述语句声明了一个名为 list 的数组，该数组包含 1000 个元素。其中每个元素都是一个 entry 结构，并可以使用下标来标识，就像其他类型的数组元素一样。这些元素都包含三个成员，其中每个成员都是一个 char 数组。图 11.3 说明了该结构数组的组织方式。

声明结构数组后，便可以以很多方式来操纵其数组。例如，要将一个数组元素的数据赋给另一个数组元素，可以这样做：

```
list[1] = list[5];
```

上述语句将结构 list[1]的每个成员的值赋给 list[5]的相应成员。也可以在结构成员之间移动数据。下面的语句：

```
strcpy(list[1].phone, list[5].phone);
```

将 list[5].phone 中的字符串复制到 list[1].phone 中（库函数 strcopy()将一个字符串复制给另一个字符串）。如果您愿意，还可以在结构的数组成员的元素之间移动数据：

```
list[5].phone[1] = list[2].phone[3];
```

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

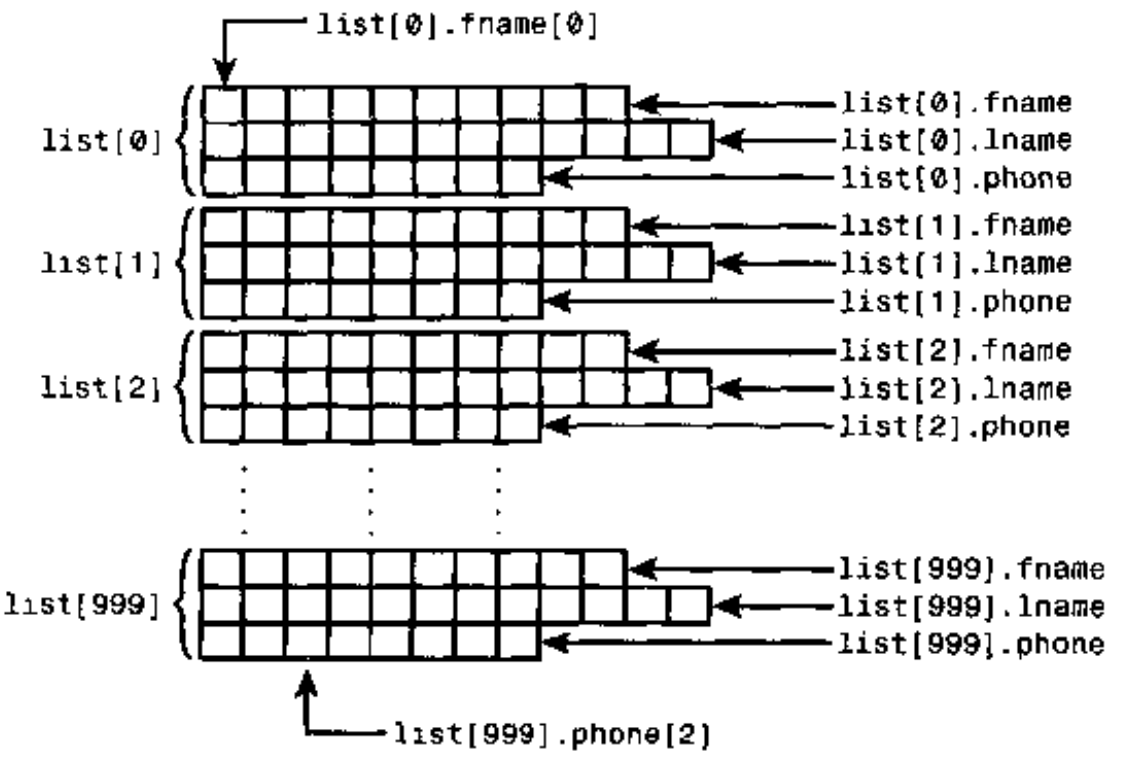


图 11.3　list 结构数组的组织方式

上述语句将 list[5]的 phone 成员的第二个字符移到 list[2]的 phone 成员的第 4 个位置（别忘了，下标从 0 开始）。

程序清单 11.4 演示了结构数组的用法，其中数组中包含的结构将数组作为其成员。

程序清单 11.4　　structarr.c：结构数组

```
1:  /* Demonstrates using arrays of structures. */
2:
3:  #include <stdio.h>
4:
5:  /* Define a structure tc hold entries. */
6:
7:  struct entry {
8:      char fname[20];
9:      char lname[20];
10:     char phone[10];
11: };
12:
13: /* Declare an array of structures. */
14:
15: struct entry list[4];
16:
17: int i;
18:
19: int main( void )
20: {
21:
22:     /* Loop to input data for four people. */
23:
24:     for (i = 0; i < 4; i++)
25:     {
26:         printf("\nEnter first name: ");
27:         scanf("%s", list[i].fname);
28:         printf("Enter last name: ");
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
29:        scanf("%s", list[i].lname);
30:        printf("Enter phone in 123-4567 format: ");
31:        scanf("%s", list[i].phone);
32:     }
33:
34:     /* Print two blank lines. */
35:
36:     printf("\n\n");
37:
38:     /* Loop to display data. */
39:
40:     for (i = 0; i < 4; i++)
41:     {
42:        printf("Name: %s %s", list[i].fname, list[i].lname);
43:        printf("\t\tPhone: %s\n", list[i].phone);
44:     }
45:
46:     return 0;
47: }
```

该程序的运行情况如下：

```
Enter first name: Bradley
Enter last name: Jones
Enter phone in 123-4567 format: 555-1212

Enter first name: Peter
Enter last name: Aitken
Enter phone in 123-4567 format: 555-3434

Enter first name: Melissa
Enter last name: Jones
Enter phone in 123-4567 format: 555-1212

Enter first name: Kyle
Enter last name: Rinni
Enter phone in 123-4567 format: 555-1234

Name: Bradley Jones          Phone: 555-1212
Name: Peter Aitken           Phone: 555-3434
Name: Melissa Jones          Phone: 555-1212
Name: Kyle Rinni             Phone: 555-1234
```

分析：该程序清单的格式与其他程序清单相同。首先是注释（第 1 行），然后是包含了标准输入/输出函数所需的头文件 stdio.h（第 3 行）。第 7～11 行定义了一个名为 entry 的结构，该结构包含三个字符数组：fname、lname 和 phone。第 15 行使用该模板定义了一个包含 4 个 entry 元素的数组：list。第 17 行定义了一个 int 变量，程序将其用作计数器。从第 19 行开始是 main()函数。它首先使用一条 for 语句来执行 4 次循环（第 24～32 行），以获取结构数组中的信息。对于 list，下标的使用方式与第 8 天课程介绍的数组变量相同。

第 36 行用于将输入和输出隔开，它打印两个空白行。第 40～44 行显示用户前面输入的数据。打印结构数组的值时，使用了带下标的数组名、成员运算符（.）和结构成员的名称。

请熟悉程序清单 11.4 中使用的技巧，很多实际的编程任务都适合使用结构数组（其中的结构又包含其他数组）来完成。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

应　该	不应该
声明以前定义过的结构的实例时，一定要使用关键字 struct； 声明结构实例时，作用域规则与其他变量相同（第 12 天的课程将完整地介绍该主题）。	使用结构的成员时，别忘了结构实例名和成员运算符（.）； 不要将结构的名称和结构实例混淆。结构名用于定义结构的模板或格式，而实例是使用结构名声明的一个变量。

11.4　初始化结构

与其他变量类型一样，也可以在声明结构时对它进行初始化，方式与初始化数组类似：在结构声明的后面加上等号和一个初始化值列表（用花括号括起，其中的值用逗号隔开）。例如，请看下面的语句：

```
1: struct sale {
2:     char customer[20];
3:     char item[20];
4:     float amount;
5: } mysale = {
6:                 "Acme Industries",
7:                 'Left-handed widget",
8:                  1000.00
9:             };
```

当上述语句执行时，将执行以下操作：

1. 定义一种名为 sale 的结构类型（第 1～5 行）；
2. 声明一个名为 mysale 的 sale 结构（第 5 行）；
3. 将结构成员 mysale.customer 初始化为字符串"Acme Industries"（第 5～6 行）；
4. 将结构成员 mysale.item 初始化为字符串“Left-handed widget”（第 7 行）；
5. 将结构成员 mysale.amount 初始化为 1000.00（第 8 行）。

对于包含其他结构的结构，应依次列出初始化值。这些值将按成员在结构定义中出现的顺序被依次赋给结构成员。下面的范例在前一个例子的基础上进行了扩展：

```
1:  struct customer {
2:      char firm[20];
3:      char contact[25];
4:  }
5:
6:  struct sale {
7:      struct customer buyer;
8:      char item[20];
9:      float amount;
10: } mysale = { { "Acme Industries", "George Adams"},
11:               "Left-handed widget",
12:                1000.00
13:             };
```

上述语句执行下述初始化工作：

1. 将结构成员 mysale.buyer.firm 初始化为字符串“Acme Industries”（第 10 行）；
2. 将结构成员 mysale.buyer.contact 初始化为字符串“George Adams”（第 10 行）；
3. 将结构成员 mysale.item 初始化为字符串“Left-handed widget”（第 11 行）；
4. 将结构成员 mysale.amount 初始化为 1000.00（第 12 行）。

也可以初始化结构数组。您提供的数组将被依次赋给数组中的结构。例如，要声明一个 sale 结构数组，并初始化其前两个数组元素（即前两个结构），可以这样编写代码：

```
1:  struct customer {
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
2:     char firm[20];
3:     char contact[25];
4: };
5:
6: struct sale {
7:     struct customer buyer;
8:     char item[20];
9:     float amount;
10: };
11:
12:
13: struct sale y1990[100] = {
14:     { { "Acme Industries", "George Adams"},
15:        "Left-handed widget",
16:         1000.00
17:     }
18:     { { "Wilson & Co.", "Ed Wilson"},
19:        "Type 12 gizmo",
20:         290.00
21:     }
22: };
```

上述代码的功能如下：

1. 将结构成员 y1990[0].buyer.firm 初始化为字符串“Acme Industries”（第 14 行）；
2. 将结构成员 y1990[0].buyer.contact 初始化为字符串“George Adams”（第 14 行）；
3. 将结构成员 y1990[0].item 初始化为字符串“Left-handed widget”（第 15 行）；
4. 将结构成员 y1990[0].amount 初始化为 1000.00（第 16 行）；
5. 将结构成员 y1990[1].buyer.firm 初始化为字符串“Wilson & Co.”（第 18 行）；
6. 将结构成员 y1990[1].buyer.contact 初始化为字符串“Ed Wilson”（第 18 行）；
7. 将结构成员 y1990[1].item 初始化为字符串“Type 12 gizmo”（第 19 行）；
8. 将结构成员 y1990[1].amount 初始化为 290.00（第 20 行）。

11.5 结构和指针

由于指针是 C 语言中非常重要的组成部分，因此指针可用于结构也就没什么可大惊小怪的了。可以将指针作为结构的成员，也可以声明指向结构的指针。接下来的几节将介绍这些主题。

11.5.1 将指针作为结构的成员

在将指针作为结构的成员方面，有很大的灵活性。指针成员的声明方式和非成员指针相同，即使用间接运算符（*）。下面是一个例子：

```
struct data
{
    int *value;
    int *rate;
} first;
```

上述语句定义并声明了一个结构，它包含两个指向 int 变量的指针成员。与所有指针一样，不仅需要声明它们，还需要将其初始化为指向某种东西。这可以通过将变量地址赋给它们来完成。如果 cost 和 interest 已经被声明为 int 变量，则可以这样编写代码：

```
first.value = &cost;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
first.rate = &interest;
```

指针被初始化后，便可以使用间接运算符（*）来获得它们指向的变量的值。表达式*first.value 指的是变量 cost 的值；而*first.rate 指的是 interest 的值。

最常被用作结构成员的指针的类型为 char。第 10 天的课程介绍过，字符串是一个字符序列，用一个指向其第一个字符的指针来表示，并用空字符来指示字符串的结尾。为复习这些内容，下面声明了一个 char 指针，并将其初始化为指向一个字符串：

```
char *p_message;
p_message = "Teach Yourself C In 21 Days";
```

也可以使用类型为 char 指针的结构成员来完成上述工作：

```
struct msg {
    char *p1;
    char *p2;
} myptrs;
myptrs.p1 = "Teach Yourself C In 21 Days";
myptrs.p2 = "By SAMS Publishing";
```

图 11.4 说明了执行上述语句后的结果。结构的每个指针成员都指向一个字符串的第一个字节，被指向的字符串存储在内存的其他地方。请将该图与图 11.3 进行比较。图 11.3 说明了包含 char 数组的结构中的数据是如何存储的。

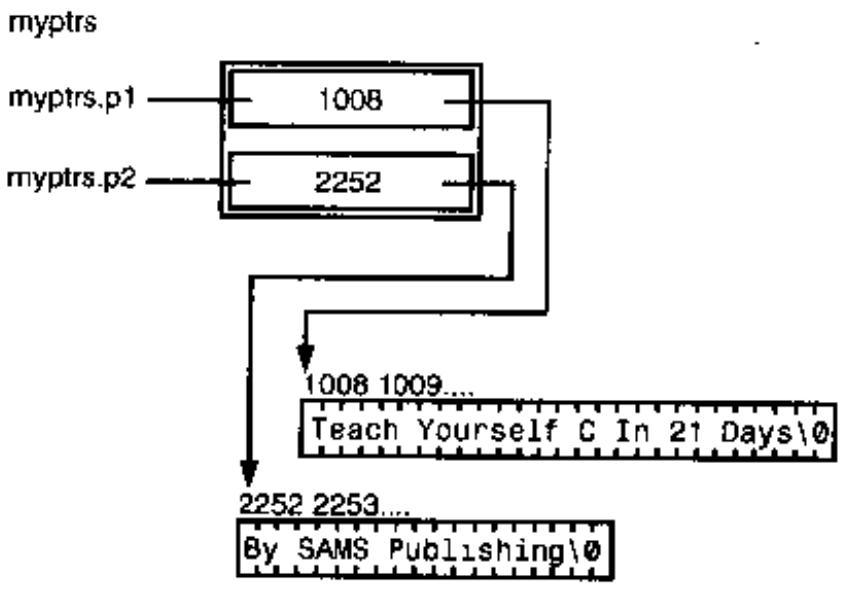


图 11.4　包含 char 指针的结构

结构的指针成员可用于任何可使用指针的地方。例如，要打印被指向的字符串，可以这样编写代码：

```
printf("%s %s", myptrs.p1, myptrs.p2);
```

将 char 数组和 char 指针作为结构成员之间有何区别呢？这是两种在结构中存储字符串的方法，下面的 msg 结构使用了这两种方法：

```
struct msg
{
    char p1[30];
    char *p2;     /* caution: uninitialized */
} myptrs;
```

您可能还记得，不带方括号的数组名是一个指针，它指向数组的第一个元素。因此，可以以类似的方式使用上述两个结构成员（注意，将值复制给 p2 之前，应初始化 p2）：

```
strcpy(myptrs.p1, "Teach Yourself C In 21 Days");
strcpy(myptrs.p2, "By SAMS Publishing");
/* additional code goes here */
puts(myptrs.p1);
puts(myptrs.p2);
```

这两种方法之间有何区别呢？区别为：如果您定义了一个包含 char 数组的结构，则该结构类型的每个实例都

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

包含了用于存储指定长度的数组的存储空间。另外，存储的字符串不能超过指定的长度。下面是一个例子：

```
struct msg
{
    char p1[10];
    char p2[10];
} myptrs;
...
strcpy(p1, "Minneapolis"); /* Wrong! String longer than array.*/
strcpy(p2, "MN");          /* Okay, but wastes space because    */
                           /* string shorter than array.        */
```

如果您定义了一种包含 char 指针的结构，则不存在上述限制。这种结构的每个实例都只包含用于存储指针的空间，实际的字符串将存储在内存的其他地方（但您无需关心它们到底存储在哪里）。既没有长度限制，也不会浪费空间。实际的字符串并没有存储在结构中，结构中的每个指针都可以指向任意长度的字符串。指针指向的字符串将是结构的一部分，虽然它们不存储在结构中。



警告：如果您不初始化指针，则可能无意间重写其他代码使用的内存。使用指针而不是数组时，请切记对指针进行初始化。这可以通过将另一个变量赋给它或动态地分配内存来完成。

11.5.2 创建指向结构的指针

在 C 程序中，可以声明并使用指向结构的指针，就像声明指向其他数据存储类型的指针一样。今天课程的后面将介绍到，将结构作为参数传递给函数时，常常使用指向结构的指针。指向结构的指针也被用于链表中（一种功能非常强大的数据存储方式），链表将在第 15 天的课程中介绍。

下面介绍如何创建并使用指向结构的指针。首先定义一种结构：

```
struct part
{
    short number;
    char name[10];
};
```

然后声明一个指向 part 结构的指针：

```
struct part *p_part;
```

上述声明中的间接运算符的含义是：p_part 是一个 part 指针，而不是一个 part 实例。

现在可以初始化该指针吗？不能，因为虽然已经定义了结构 part，但还没有声明这种结构的实例。别忘了，是声明而不是定义为数据对象预留了内存空间。由于指针需要指向内存地址，因此您必须声明一个 part 实例，指针才有东西可指。声明如下：

```
struct part gizmo;
```

现在，便可以初始化指针了：

```
p_part = &gizmo;
```

上述语句将 gizmo 的地址赋给 p_part。图 11.5 说明了结构和指向它的指针之间的关系。

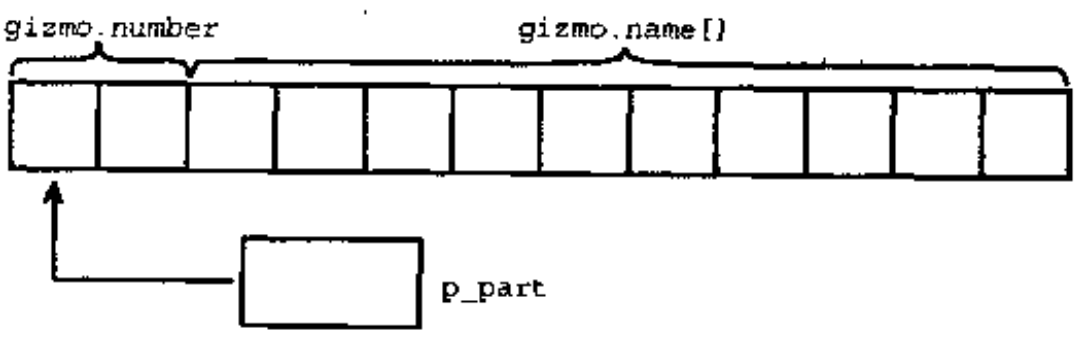


图 11.5 指向结构的指针指向结构的第一个字节

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

有了一个指向结构 gizmo 的指针后，如何使用它呢？一种方法是使用间接运算符（*）。第 9 天的课程介绍过，如果 ptr 是指向一个数据对象的指针，则表达式*ptr 是被指向的对象。

将这一规则用于这个范例。因为 p_part 是指向结构 gizmo 的指针，所以*p_part 是 gizmo。这样便可以使用结构成员运算符（.）来存取 gizmo 的各个成员。要将 100 赋给 gizmo.number，可以这样编写代码：

```
(*p_part).number = 100;
```

由于结构成员运算符（.）的优先级比间接运算符（*）高，因此必须用圆括号将*p_part 括起。

另一种通过指针来存取结构成员的方法是使用间接成员运算符(->)，间接成员运算符由字符-和>组成(注意，当这两个符号在一起时，编译器将其视为一个运算符，而不是两个)。该运算符用于连接指针名和成员名。例如，要通过指针 p_part 来存取结构 gizmo 的 number 成员，可以这样编写代码：

```
p_part->number
```

请看另一个例子。如果 str 是一个结构，p_str 是指向 str 的指针，而 memb 是 str 的一个成员，则可以通过下面的代码来存取 str.memb：

```
p_str->memb
```

因此，有三种存取结构成员的方式：

- 通过结构名；
- 通过指向结构的指针和间接运算符（*）；
- 通过指向结构的指针和间接成员运算符（->）。

如果 p_str 是指向结构 str 的指针，则下面的三个表达式是等价的：

```
str.memb
(*p_str).memb
p_str->memb
```



> 注意：有些人将间接成员运算符叫做“结构指针运算符”。

11.5.3　使用指针和结构数组

前面介绍过，结构数组是一个功能强大的编程工具，指向结构的指针也是如此。可以把两者结合起来使用，通过指针来存取数组中的结构元素。

前面的一个范例中，定义了一个这样的结构：

```
struct part
{
    short number;
    char name[10];
};
```

定义 part 结构后，便可以声明一个 part 数组。

```
struct part data[10];
```

接下来，声明一个 part 指针，并将其初始化为指向数组 data 中的第一个结构：

```
struct part *p_part;
p_part = &data[0];
```

以前介绍过，不带方括号的数组名是一个指针，它指向数组的第一个元素，因此上面的第 2 行代码也可以写成：

```
p_part = data;
```

现在，有一个由 part 结构组成的数组和一个指向第一个数组元素（即数组中的第一个结构）的指针。可以使用下面的语句来打印第一个元素的内容：

```
printf("%d %s", p_part->number, p_part->name);
```

如果要打印所有的数组元素，该如何办呢？可以使用一个 for 循环，在每次迭代中打印一个数组元素。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

要通过指针表示法来存取成员，必须修改 p_part，使之在循环的每一次迭代后，指向下一个数组元素（即数组中的下一个结构）。如何修改呢？

可以使用指针算术。被用于指针时，单目递增运算符（++）有特殊的含义：将指针加上它指向的对象的长度。换句话说，如果指针 ptr 指向一个 obj 类型的数据对象，则语句：

```
ptr++;
```

与下面的语句等价：

```
ptr += sizeof(obj);
```

指针算术的这种特征对数组非常适用，因为数组元素在内存中是顺序存储的。如果指针指向第 n 个元素，则使用运算符++将指针加 1 后，它将指向第 n+1 个元素。图 11.6 说明了这一点，该图说明了一个名为 x[]的数组，其中的每个元素占 4 字节（例如，一个包含两个 short 成员的结构，每个成员占两个字节）。指针 ptr 被初始化为指向 x[0]，ptr 每被递增一次，便指向接下来的一个数组元素。

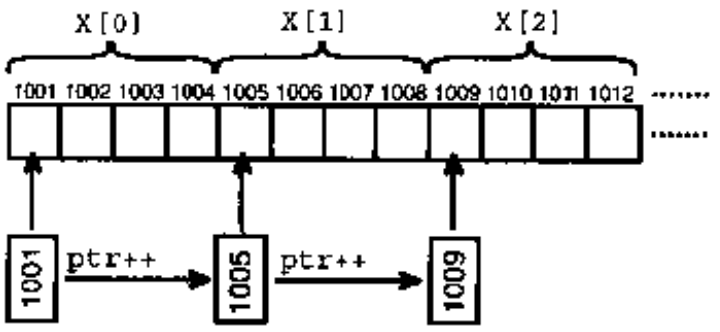


图 11.6 每被递增一次，指针都将指向接下来的一个数组元素

这意味着程序可以通过递增指针来遍历结构数组（或其他任何数据类型的数组）。与使用数组下标相比，这种表示法通常更容易使用，也更简洁。程序清单 11.5 演示了如何完成这样的工作。

程序清单 11.5　　access.c：通过递增指针来存取后续的元素

```
1:  /* Demonstrates stepping through an array of structures */
2:  /* using pointer notation. */
3:
4:  #include <stdio.h>
5:
6:  #define MAX 4
7:
8:  /* Define a structure, then declare and initialize */
9:  /* an array of 4 structures. */
10:
11: struct part {
12:     short number;
13:     char name[10];
14: } data[MAX] = {1, "Smith",
15:                2, "Jones",
16:                3, "Adams",
17:                4, "Wilson"
18:               };
19:
20: /* Declare a pointer to type part, and a counter variable. */
21:
22: struct part *p_part;
23: int count;
24:
25: int main( void )
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
26: {
27:     /* Initialize the pointer to the first array element. */
28:
29:     p_part = data;
30:
31:     /* Loop through the array, incrementing the pointer */
32:     /* with each iteration. */
33:
34:     for (count = 0; count < MAX; count++)
35:     {
36:         printf("At address %d: %d %s\n", p_part, p_part->number,
37:                 p_part->name);
38:         p_part++;
39:     }
40:
41:     return 0;
42: }
```

该程序的输出如下：

```
At address 4202504: 1 Smith
At address 4202516: 2 Jones
At address 4202528: 3 Adams
At address 4202540: 4 Wilson
```

分析：首先，在第 11～18 行，程序声明并初始化了一个由 part 结构组成的名为 data 的数组。然后第 22 行定义了一个名为 p_part 的指针，用于指向 data 数组中的第一个结构。main()函数首先将 p_part 指向前面声明的 data 数组中的第一个 part 结构（第 29 行）。然后，使用一个 for 循环（第 34～39 行）来打印所有的元素，该循环每次迭代时，都将指向数组的指针递增。同时，该程序还显示了每个元素的地址。

在您的计算机上运行该程序时，显示的地址可能与上面不同，但地址之间的差值应该相同——等于结构 part 的长度。这清晰地表明，将指针递增时，增加的量为它指向的数据对象的长度。

11.5.4　将结构作为参数传递给函数

与其他数据类型一样，也可以将结构作为参数传递给函数。程序清单 11.6 演示了如何做。该程序是对程序清单 11.3 进行修改后得到的，它使用一个函数将传递过来的结构中的信息显示到屏幕上，而程序清单 11.3 是直接在 main()中进行显示的。

程序清单 11.6　　func.c：将结构作为参数传递给函数

```
1:  /* Demonstrates passing a structure to a function. */
2:
3:  #include <stdio.h>
4:
5:  /* Declare and define a structure to hold the data. */
6:
7:  struct data {
8:      float amount;
9:      char fname[30];
10:     char lname[30];
11: } rec;
12:
13: /* The function prototype. The function has no return value, */
14: /* and it takes a structure of type data as its one argument. */
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
15:
16: void print_rec(struct data diplayRec);
17:
18: int main( void )
19: {
20:     /* Input the data from the keyboard. */
21:
22:     printf("Enter the donor's first and last names,\n");
23:     printf("separated by a space: ");
24:     scanf("%s %s", rec.fname, rec.lname);
25:
26:     printf("\nEnter the donation amount: ");
27:     scanf("%f", &rec.amount);
28:
29:     /* Call the display function. */
30:     print_rec( rec );
31:
32:     return 0;
33: }
34: void print_rec(struct data displayRec)
35: {
36:     printf("\nDonor %s %s gave $%.2f.\n", displayRec.fname,
37:            displayRec.lname, displayRec.amount);
38: }
```

该程序的运行情况如下：

```
Enter the donor's first and last names,
separated by a space: Bradley Jones

Enter the donation amount: 1000.00

Donor Bradley Jones gave $1000.00.
```

分析：第16行是函数的原型，该函数将结构作为参数。与传递其他数据类型的参数一样，必须传递合适的参数。在这里，参数为data类型的结构。

第34行的函数头再次说明了这一点。调用该函数时，只需传递结构实例名，这里为rec（第30行）。将结构作为参数传递给函数与传递简单变量没有什么不同。

也可以传递地址（即指向结构的指针）将结构传递给函数。事实上，在老式的C语言版本中，只能以这样的方式将结构传递给函数。现在，情况并非如此，但老式程序使用的是这种方法。传递指针参数时，在函数中必须使用间接成员运算符（->）来存取结构的成员。

应　该	不应该
一定要利用指向结构的指针，尤其是使用结构数组时。	不要将数组和结构混淆。
通过指向结构的指针来存取成员时，一定要使用间接成员运算符（->）。	请别忘了，将指针递增时，移动的距离等于它指向的数据的长度。对于指向结构的指针，为结构的长度。

11.6　共用体

共用体类似于结构，其声明和使用方式与结构相同。共用体和结构之间的区别在于，同一时间内，只有一个成员可用。原因很简单，共用体的所有成员占用的是同一个内存区域——它们相互覆盖。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

11.6.1　定义、声明和初始化共用体

共用体的定义和声明方式与结构相同。唯一的区别在于，声明时使用关键字 union，而不是 struct。要定义一个由 char 变量和 int 变量组成的共用体，可以这样编写代码：

```
union shared
{
    char c;
    int i;
};
```

该共用体（shared）可用来创建这样的共用体实例，即存储一个字符值（c）或整型值（i）。这是一个 OR 关系。结构可同时存储多个值，而共用体只能存储一个值。图 11.7 说明了共用体 shared 在内存中的情况。

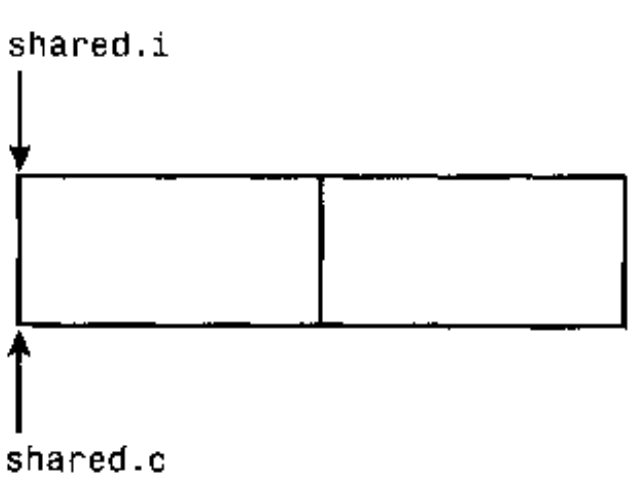


图 11.7　共用体只能同时存储一个值

可以在声明时对共用体进行初始化。由于同一时间内只有一个成员可用，因此只能初始化一个成员。为避免混淆，只允许初始化共用体的第一个成员。下面的代码声明并初始化了共用体 shared 的一个实例：

```
union shared generic_variable = {'@'};
```

上面初始化共用体 generic_variable 时，就像只初始化结构的第一个成员一样。

11.6.2　存取共用体的成员

使用共用体成员的方式与使用结构成员相同——通过成员运算符（.），但有一个很重要的区别，即同一时间内，只能存取共用体的一个成员。由于共用体存储其成员时将相互覆盖，因此每次只能存取一个成员，这非常重要。程序清单 11.7 是一个错误地使用共用体的范例。

程序清单 11.7　　union.c：一个错误地使用共用体的范例

```
1:  /* Example of using more than one union member at a time */
2:  #include <stdio.h>
3:
4:  int main( void )
5:  {
6:      union shared_tag {
7:          char   c;
8:          int    i;
9:          long   l;
10:         float  f;
11:         double d;
12:     } shared;
13:
14:     shared.c = '$';
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
15:
16:     printf("\nchar c   = %c",  shared.c);
17:     printf("\nint i    = %d",  shared.i);
18:     printf("\nlong l   = %ld", shared.l);
19:     printf("\nfloat f  = %f",  shared.f);
20:     printf("\ndouble d = %f",  shared.d);
21:
22:     shared.d = 123456789.8765;
23:
24:     printf("\n\nchar c   = %c",  shared.c);
25:     printf("\nint i    = %d",  shared.i);
26:     printf("\nlong l   = %ld", shared.l);
27:     printf("\nfloat f  = %f",  shared.f);
28:     printf("\ndouble d = %f\n",  shared.d);
29:
30:     return 0;
31: }
```

该程序的输出如下：

```
char c   = $
int i    = 65572
long l   = 65572
float f  = 0.000000
double d = 0.000000

char c   = 7
int i    = 1468107063
long l   = 1468107063
float f  = 284852666499072.000000
double d = 123456789.876500
```

分析：第6～12行定义并声明了一个名为shared的共用体。shared包含5个成员，其中每个成员的类型各不相同。第14和22行初始化了shared的成员。然后，第16～20行和第24～28行使用printf()语句打印每个成员的值。

注意，在您的计算机上运行该程序时，除了char c = $和double d = 123456789.876500外，其他输出可能不同。由于第14行初始化了字符变量c，因此在初始化其他成员之前，只应使用该成员。此时如果打印共用体的其他成员（第16～20行），结果将是不确定的。第22行给double成员d赋值，因此此时打印各个成员时，除d外，其他成员的值都是不确定的。第14行赋给c的值已经丢失，因为第22行赋给d的值将c的值覆盖了。这表明，不同成员被存储在相同的内存空间中。

定义和声明共用体的语法如下：

```
union tag {
    union_member(s);
    /* additional statements may go here */
}instance;
```

关键字union用于声明共用体。共用体是一组变量，其中各个成员被存储在相同的内存区域中。

关键字union用于标识共用体定义的开始，接着是提供给共用体的名称。名称后面是共用体的成员，用花括号括起。定义共用体的同时，可以声明一个实例。如果定义共用体时，没有声明实例，则共用体将作为一个模板，供程序以后用来声明共用体。模板的格式如下：

```
union tag {
    union_member(s);
    /* additional statements may go here */
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
};
```

使用模板来声明共用体的格式如下：

```
union tag instance;
```

下面是 3 个定义和声明共用体的范例：

范例 1：

```
/* Declare a union template called tag */
union tag {
    int nbr;
    char character;
}
/* Use the union template */
union tag mixed_variable;
```

范例 2：

```
/* Declare a union and instance together */
union generic_type_tag {
    char c;
    int i;
    float f;
    double d;
} generic;
```

范例 3：

```
/* Initialize a union. */
union date_tag {
    char full_date[9];
    struct part_date_tag {
         char month[2];
         char break_value1;
         char day[2];
         char break_value2;
         char year[2];
    } part_date;
}date = {"01/01/97"};
```

程序清单 11.8 演示了一种比较实用的共用体用法。虽然这种用法很简单，但这是比较常见的一种用法。

程序清单 11.8　　unicm2.c：一种实用的共用体用法

```
1:  /* Example of a typical use of a union */
2:
3:  #include <stdio.h>
4:
5:  #define CHARACTER   'C'
6:  #define INTEGER     'I'
7:  #define FLOAT       'F'
8:
9:  struct generic_tag{
10:     char type;
11:     union shared_tag {
12:         char   c;
13:         int    i;
14:         float  f;
15:     } shared;
16: };
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
17:
18: void print_function( struct generic_tag generic );
19:
20: int main( void )
21: {
22:     struct generic_tag var;
23:
24:     var.type = CHARACTER;
25:     var.shared.c = '$';
26:     print_function( var );
27:
28:     var.type = FLOAT;
29:     var.shared.f = (float) 12345.67890;
30:     print_function( var );
31:
32:     var.type = 'x';
33:     var.shared.i = 111;
34:     print_function( var );
35:     return 0;
36: }
37: void print_function( struct generic_tag generic )
38: {
39:     printf("\n\nThe generic value is...");
40:     switch( generic.type )
41:     {
42:        case CHARACTER: printf("%c",  generic.shared.c);
43:                        break;
44:        case INTEGER:   printf("%d",  generic.shared.i);
45:                       break;
46:        case FLOAT:     printf("%f",  generic.shared.f);
47:                        break;
48:        default:       printf("an unknown type: %c\n",
49:                            generic.type);
50:                       break;
51:     }
52: }
```

该程序的输出如下：

```
The generic value is...$

The generic value is...12345.678711

The generic value is...an unknown type: x
```

分析：该程序很简单，它说明了如何使用共用体。

该程序提供了一种使用相同的存储空间来存储多种数据类型的方式。结构 generic_tag 使您能够在相同的区域内存储一个字符、一个整数或一个浮点数。该区域是一个名为 shared 的共用体，其工作原理与程序清单 11.7 相同。结构 generic_tag 还包含一个名为 type 的成员，用于指出 shared 中包含的变量类型，以防止错误地使用 shared，从而避免程序清单 11.7 那样的错误数据。

第 5、6 和 7 行分别定义了常量 CHARACTER、INTEGER 和 FLOAT，程序使用这些常量来提高可读性。第 9～16 行定义了一个 generic_tag 结构，供后面使用。第 18 行是函数 print_function()的原型。第 22 行声明

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

了一个名为 var 的结构，然后第 24 和 25 行将其初始化为存储一个字符。第 26 行调用函数 print_function()来打印共用体中存储的值。第 28～30 行和第 32～34 行分别存储并打印其他两种类型的值。

该程序清单的核心是 print_function()函数。虽然该函数用于打印 generic_tag 变量的值，但也可以编写一个类似的函数来初始化它的值。print_function()函数检查 type 变量的值，以执行合适的打印语句。这样可以避免出现程序清单 11.7 那样的错误数据。

应　该	不应该
请切记当前使用的是共用体的哪个成员。如果您给一个成员赋值后，使用另一个成员，结果将出乎意料。	不要试图初始化共用体中除第一个成员之外的成员。 别忘了，共用体的长度等于其最长成员的长度。

11.7　使用 typedef 给结构创建别名

可以使用关键字 typedef 给结构或共用体类型创建别名。例如，下面的语句将结构的别名定义为 coord：

```
typedef struct {
    int x;
    int y;
} coord;
```

然后，便可以使用标识符 coord 来声明这种结构的实例：

```
coord topleft, bottomright;
```

注意，typedef 不同于结构名。编写下面的代码后：

```
struct coord {
    int x;
    int y;
};
```

标识符 coord 是该结构的名称。可以使用该名称来声明这种结构的实例，但与使用 typedef 不同，必须包含关键字 struct：

```
struct coord topleft, bottomright;
```

使用 typedef 和结构名之间的差别不大。使用 typedef 时，要简洁些，因此无需包含关键字 struct。另一方面，使用结构名和关键字 struct 时，能清晰地说明声明的是一个结构。

11.8　总　结

今天的课程介绍了如何使用结构——一种为满足程序需求而设计的数据类型。结构可以包含任何数据类型，包括其他结构、指针和数组。结构中的数据项被称为成员。要存取成员，可使用结构名和成员名，并用结构成员运算符（.）将它们连接起来。可以使用单个结构，也可以使用结构数组。

共用体与结构类似，它们之间的主要区别在于，共用体在同一个区域内存储其所有的成员。这意味着只能同时使用共用体的一个成员。

11.9　问与答

问：有什么原因使得定义结构时，不应同时声明实例吗？

答：今天的课程介绍了两种定义结构的方式。第一种是在定义结构体和结构名的同时，声明结构实例；另一种是定义结构体和结构名时，不声明结构实例，以后再使用关键字 struct、结构名、实例名来声明实例。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

常用的方式是第二种。很多程序员在定义结构体和结构名时，不声明任何结构实体，而是以后再声明结构实体。明天的课程将介绍变量的作用域，这些作用域规则适用于实例，但对结构名和结构体不起作用。

问：人们更常用 typedef 还是结构名？

答：很多程序员使用 typedef 来提高代码的可读性，但实际上的差别不大。市面上有很多包含函数的插件库，这些插件（尤其是数据库插件）通常包含大量的 typedef，以使产品是独特的。

问：可以使用赋值运算符将一个结构赋给另一个吗？

答：较新的 C 编译器允许这样做，但老版本不允许。对于老版本的 C 编译器，必须分别给结构的成员赋值。共用体的情况也是如此。

问：共用体的长度是如何确定的？

答：由于共用体的每个成员都被存储在同一个内存单元中，因此存储共用体所需的空间为其最大成员的长度。

11.10 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

11.10.1 小测验

1. 结构和数组之间有何区别？
2. 何为结构成员运算符？它有何用途？
3. 创建结构时使用哪个关键字？
4. 结构名和结构实例之间有何区别？
5. 下述代码片段完成什么工作？

```
struct address
{
    char name[31];
    char add1[31];
    char add2[31];
    char city[11];
    char state[3];
    char zip[11];
} myaddress = { "Bradley Jones",
                "RTSoftware",
                "P.O. Box 1213",
                "Carmel", "IN", "46082-1213"};
```

6. 如果您创建了一个名为 word 的 typedef，如何使用它来定义一个名为 my Word 的变量？
7. 假设您声明了一个结构数组，而 ptr 是一个指针，它指向该数组的第一个元素（即数组中的第一个结构）。如何修改 ptr，使之指向第二个数组元素？

11.10.2 练习

1. 定义一个名为 time 的结构，它包含三个 int 成员。
2. 请编写执行下述两项任务的代码：定义一个名为 data 的结构，它包含一个 int 成员和两个 float 成员；声明一个名为 info 的 data 结构实例。
3. 继续练习 2，将 100 赋给结构 info 的 int 成员。
4. 声明一个指针，并将其初始化为指向 info。
5. 继续练习 4，以两种不同的方式，使用指针表示法将 5.5 赋给 info 的第一个 float 成员。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

6. 定义一种名为 data 的结构类型，它包含一个字符串，该字符串最长为 20 个字符。

7. 定义一种包含 5 个字符串的结构：address1、address2、city、state 和 zip；然后创建一个名为 RECORD 的 typedef，该 typedef 可用于声明这种结构的实例。

8. 使用练习 7 中创建的 typedef，声明并初始化一个名为 myaddress 的结构实例。

9. 排错：下面的代码有何错误？

```
struct {
    char zodiac_sign[21];
    int month;
} sign = "Leo", 8;
```

10. 排错：下面的代码有何错误？

```
/* setting up a union */
union data{
    char a_word[4];
    long a_number;
}generic_variable = { "WOW", 1000 };
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第 12 天课程　变量作用域

第 5 天的课程介绍过，在函数内部定义的变量不同于函数外部定义的变量。这其实介绍了一个概念：变量作用域，只是您不知道而已。变量作用域是 C 语言编程中非常重要的一个方面。今天将介绍以下内容：

- 变量作用域的概念及其重要的原因；
- 何为外部变量？为何要避免使用这种变量？
- 有关局部变量的方方面面；
- 静态变量和动态变量之间的区别；
- 局部变量和代码块；
- 如何选择存储类型（storage class）？

12.1　作用域是什么

变量的作用域指的是程序中的哪些部分可以访问变量，换句话说，在哪些地方，变量是可见的。对于变量而言，可访问性和可见性的含义相同。说到作用域时，变量指的是所有数据类型的变量：简单变量、结构、数组、指针等，也可能指的是用关键字 const 定义的符号常量。

作用域还会影响变量的生命周期——变量在内存中存活的时间，或者说什么时候分配和释放变量的存储空间。今天的课程首先简要地介绍作用域，然后详细地探讨可见性和作用域。

12.1.1　演示作用域

请看程序清单 12.1 中的程序。第 5 行定义了一个名为 x 的变量，然后第 11 行使用 printf()显示 x 的值。接下来调用函数 print_value()再次显示 x 的值。注意，并没有将 x 作为参数传递给函数 print_value()，该函数将 x 作为参数传递给 printf()（第 19 行）。

程序清单 12.1　　scope.c：在函数 print_value()中，可以访问变量 x

```
1:  /* Illustrates variable scope. */
2:
3:  #include <stdio.h>
4:
5:  int x = 999;
6:
7:  void print_value(void);
8:
9:  int main( void )
10: {
11:     printf("%d\n", x);
12:     print_value();
13:
```

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
14:     return 0;
15: }
16:
17: void print_value(void)
18: {
19:         printf("%d\n", x);
20: }
```

该程序的输出如下：

```
999
999
```

该程序在编译和运行时没有任何问题。现在，对该程序稍做修改，将变量 x 的声明移到 main()函数内部。修改后的源代码如程序清单 12.2 所示，其中 x 的声明位于第 9 行。

程序清单 12.2　　scope2.c：在函数 print_value()中，不能访问变量 x

```
1:  /* Illustrates variable scope. */
2:
3:  #include <stdio.h>
4:
5:  void print_value(void);
6:
7:  int main( void )
8:  {
9:     int x = 999;
10:
11:    printf("%d\n", x);
12:    print_value();
13:
14:    return 0;
15: }
16:
17: void print_value(void)
18: {
19:    printf("%d\n", x);
20: }
```

分析：如果编译程序清单 12.2，编译器将生成一条与下面类似的错误消息：

```
list1202.c(19) : Error: undefined identifier 'x'
```

以前介绍过，在错误消息中，用圆括号括起的编号指的是有错误的程序行。第 19 行位于函数 print_value()中，它调用 printf()。

该错误消息指出，编译位于 print_value()函数中的第 19 行时，变量 x 没有被定义，或者说不可见。然而，第 11 行调用 printf()时，并没有产生错误消息，这说明在 print_value()的外面，变量 x 是可见的。

程序清单 12.1 和 12.2 之间的唯一差别是，定义变量 x 的位置不同。通过移动 x 的声明，您改变了它的作用域。在程序清单 12.1 中，x 是在 main()函数外面被定义的，因此它是一个外部变量，其作用域为整个程序，无论是在 main()函数还是 print_value()函数中，它都是可访问的。在程序清单 12.2 中，x 是在函数 main()的内面定义的，因此是一个局部变量，其作用域为 main()函数内。在 print_value()看来，x 并不存在，这就是编译器产生错误消息的原因所在。今天课程的后面将介绍外部变量和局部变量，但首先来介绍作用域的重要性。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

12.1.2 作用域为何重要

为理解作用域的重要性，有必要复习一下第 5 天课程中关于结构化编程的讨论。您可能还记得，结构化编程将程序划分为彼此独立的函数，其中每个函数完成一项特定的任务。这里的重点是独立。要真正独立，每个函数的变量必须不受其他函数中代码的影响。只有将每个函数的数据隔离，才能确保函数在完成其工作时，不会遭到程序其他部分的破坏。通过在函数中定义变量，可以对程序的其他部分隐藏这些变量。

然而，并非总是需要在函数之间将数据完全隔离。稍后您将知道，通过指定变量的作用域，程序员能够很好地控制数据的隔离程度。

12.2 外部变量

外部变量是在函数外面定义的变量。这意味着也在 main()的外面，因为 main()也是一个函数。到目前为止，本书定义的大部分变量都是外部变量，它们位于 main()函数之前。外部变量有时候也叫全局变量。



> 注意：如果您没有显式地初始化外部变量（给它赋值），那么在定义外部变量时，编译器将把它初始化为 0。

12.2.1 外部变量的作用域

外部变量的作用域为整个程序。这意味着外部变量在 main()函数以及整个程序中的其他任何函数中都是可见的。例如，程序清单 12.1 中的 x 是一个外部变量。正如您在编译和运行该程序时看到的，x 在函数 main()和 print_value()中都是可见的，如果您在该程序中添加其他函数，则 x 在这些函数中也是可见的。

严格地讲，说外部变量的作用域为整个程序并不准确，而应该说外部变量的作用域为包含该变量定义的整个源代码文件。如果整个程序被包含在一个源代码文件中，则这两种有关作用域的说法是等价的。大多数中小型 C 程序都被包含在一个文件中，您现在编写的程序也是如此。

然而，程序的源代码可能包含在多个独立文件中。第 21 天将介绍为何以及如何这样做，那时您将知道在这种情况下，需要对外部变量做什么样的特殊处理。

12.2.2 何时使用外部变量

虽然到目前为止，所有的范例程序都使用了外部变量，但实际编程时，您很少使用外部变量。为什么呢？因为使用外部变量违背了模块独立的原则，而这种原则是结构化编程的核心。模块独立指的是程序中的每个函数（或模块）都包含完成其任务所需的所有代码和数据。对于相对较小的程序（您现在编写的就是），这可能不那么重要，但当程序更大、更复杂时，过分地依赖于外部变量将引发问题。

何时使用外部变量呢？仅当程序的所有或大部分函数都需要使用某个变量，才应将其声明为外部变量。通常，使用 const 定义的符号常量应是外部的。如果只有一些函数需要访问某个变量，则应将该变量作为参数传递给这些函数，并不应将它声明为外部变量。

12.2.3 extern 关键字

当函数使用外部变量时，在该函数内部使用关键字 extern 来声明该变量。这是一种良好的编程惯例。声明的格式如下：

```
extern type name;
```

其中 type 是变量的类型，name 是变量的名称。例如，可以在程序清单 12.1 中的 main()和 print_value()函数中添加变量 x 的声明。添加后的代码如程序清单 12.3 所示。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 12.3　　extern.c：在函数 main()和 print_value()中将外部变量 x 声明为 extern

```
1:  /* Illustrates declaring external variables. */
2:
3:  #include <stdio.h>
4:
5:  int x = 999;
6:
7:  void print_value(void);
8:
9:  int main( void )
10: {
11:     extern int x;
12:
13:     printf("%d\n", x);
14:     print_value();
15:
16:     return 0;
17: }
18:
19: void print_value(void)
20: {
21:     extern int x;
22:     printf("%d\n", x);
23: }
```

该程序的输出如下：

```
999
999
```

该程序打印变量 x 的值两次，首先是在第 13 行（main()函数内），然后是在第 22 行（print_value()函数内）。第 5 行将 x 定义为一个值为 999 的 int 变量。第 11 和 21 行将 x 声明为 extern int。变量定义和 extern 声明是不同的，前者为变量预留存储空间，后者则指出“该函数使用了一个在其他地方定义的外部变量，其名称是什么，类型是什么。”在这个例子中，extern 声明不是必不可少的，严格地说，没有第 11 和 21 行，程序照样能正确运行。然而，如果 print_value()和变量 x 的全局声明（第 5 行）位于不同的代码模块中，则 extern 声明是必不可少的。

如果您删除第 5 行，该程序仍可通过编译，但运行时将出错。这是因为这些函数期望 x 在其他地方被定义。

12.3　局部变量

局部变量是在函数内部定义的变量，其作用域为它所在的函数。第 5 天的课程介绍了函数内的局部变量、如何使用它们以及它们的优点。编译器不会自动将局部变量初始化为 0，如果您在定义局部变量时没有对它进行初始化，则它的值将是不确定的。使用局部变量之前，必须明确地给它赋值。

main()内也可以有局部变量，程序清单 12.2 中的 x 就属于这种情况。该变量是在 main()中定义的，正如编译该程序时表明的，它只在 main()内可见。

应　该	不应该
对于诸如循环计数器等，一定要使用局部变量。 一定要使用局部变量来将变量的值与程序的其他部分隔离开来。	对于并非程序的大部分函数都需要访问的变量，不应将其声明为外部变量。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

12.3.1 静态变量和动态变量

默认情况下，局部变量是动态的。这意味着每次调用函数时，都要重新创建变量；而当函数执行完毕后，变量将被释放。实际上，这意味着在函数被两次调用之间，函数中的动态变量的值将丢失。

假设程序中有一个函数使用了一个名为 x 的局部变量，当程序首次调用该函数时，将 100 赋给了 x。稍后，程序再次调用该函数，此时变量 x 的值仍为 100 吗？不是。首次调用该函数时，当该函数执行完毕后，变量 x 的实例被释放。当函数再次被调用时，将新建一个变量 x 的实例，而原来的 x 已经不存在了。

如果想在两次调用函数之间保留局部变量的值，该如何办呢？例如，一个打印函数可能需要记住已经给打印机发送了多少行，以决定何时需要换页。为在两次调用函数之间保留局部变量的值，必须使用关键字 static 将它定义为静态的。例如：

```
void print(int x)
{
     static int lineCount;
     /* Additional code goes here */
}
```

程序清单 12.4 说明了动态局部变量和静态局部变量之间的区别。

程序清单 12.4　　static.c：静态局部变量和动态局部变量之间的区别

```
1:  /* Demonstrates automatic and static local variables. */
2:  #include <stdio.h>
3:  void func1(void);
4:  int main( void )
5:  {
6:     int count;
7:
8:     for (count = 0; count < 20; count++)
9:     {
10:        printf("At iteration %d: ", count);
11:        func1();
12:     }
13:
14:     return 0;
15: }
16:
17: void func1(void)
18: {
19:     static int x = 0;
20:     int y = 0;
21:
22:     printf("x = %d, y = %d\n", x++, y++);
23: }
```

该程序的输出如下：

```
At iteration 0: x = 0, y = 0
At iteration 1: x = 1, y = 0
At iteration 2: x = 2, y = 0
At iteration 3: x = 3, y = 0
At iteration 4: x = 4, y = 0
At iteration 5: x = 5, y = 0
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
At iteration 6: x = 6, y = 0
At iteration 7: x = 7, y = 0
At iteration 8: x = 8, y = 0
At iteration 9: x = 9, y = 0
At iteration 10: x = 10, y = 0
At iteration 11: x = 11, y = 0
At iteration 12: x = 12, y = 0
At iteration 13: x = 13, y = 0
At iteration 14: x = 14, y = 0
At iteration 15: x = 15, y = 0
At iteration 16: x = 16, y = 0
At iteration 17: x = 17, y = 0
At iteration 18: x = 18, y = 0
At iteration 19: x = 19, y = 0
```

分析：该程序有一个名为 func1()的函数，其中定义并初始化了一个静态局部变量和一个动态局部变量。该函数位于第 17～23 行。每当该函数被调用时，都将两个变量显示到屏幕上，并将其值加 1（第 22 行）。位于第 4～15 行的 main()函数包含一个 for 循环（第 8～12 行），该循环一共迭代 20 次，每次迭代打印一条消息（第 10 行）并调用函数 func1()（第 11 行）。

从输出中可知，每次迭代后，静态变量 x 的值都加 1，因为在两次调用函数之间，该变量的值保持不变；而每次调用函数时，动态变量 y 都被初始化为 0，因此其值没有递增。

该程序还表明，显式初始化（即在定义变量时对它进行初始化）的处理方式也不同。静态变量只在函数首次被调用时初始化一次，以后调用时，程序知道该变量已被初始化，因此不会再次初始化，而变量仍为前一次退出函数时的值。而对于动态变量，每次函数被调用时，都会被初始化为指定的值。

在您试验使用动态变量时，情况可能与上面说的不同。例如，如果修改程序清单 12.4，使之在定义这两个局部变量时不对它们进行初始化，即将第 17～23 行的函数 func1()修改成如下所示：

```
17: void func1(void)
18: {
19:     static int x;
20:     int y;
21:
22:     printf("x = %d, y = %d\n", x++, y++);
23: }
```

当您运行修改后的程序时，可能发现每次迭代后，y 的值都增加 1。这意味着在两次调用函数之间，y 的值保留不变，虽然它是一个动态局部变量。上面有关动态变量的讨论是错误的吗？

不是。如果您发现两次调用函数之间，动态变量的值保持不变，这只是一种巧合。出现这种情况的原因如下：每次函数被调用时，都创建一个新的 y，但在两次调用时，编译器用来存储 y 的内存单元是相同的。如果函数没有显式地初始化 y，则该内存单元仍保留 y 在前一次调用期间的值。看起来好像变量的值保持不变，但这只是一种巧合而已；并非总会出现这样的情况。

由于默认情况下，局部变量是动态的，因此无需在变量定义中指定这一点。如果您愿意，也可以在定义中加上关键字 auto，并将他放在类型关键字之前，如下所示：

```
void func1(int y)
{
   auto int count;
   /* Additional code goes here */
}
```

12.3.2　函数参数的作用域

函数头中的参数列表中定义的变量的作用域为局部的。例如，在下面的函数中：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
void func1(int x)
{
    int y;
    /* Additional code goes here */
}
```

x 和 y 都是局部变量，其作用域为整个 func1()函数。当然，x 的初始值为调用程序传递给该函数的值。您可以像使用其他局部变量那样使用 x。

由于参数变量的初始值总是为传递的相应实参的值，因此，对它们来说，动态或静态的概念毫无意义。

12.3.3 外部静态变量

可以使用关键字 static 将外部变量声明为静态的：

```
static float rate;

int main( void )
{
    /* Additional code goes here */
}
```

静态外部变量和常规外部变量之间的区别在于作用域。常规外部变量对于其所在文件中的所有函数而言都是可见的，同时其他文件中的函数也可以使用它；而静态外部变量只位于它所在的文件中，且在它的定义之后的函数中可见。

仅当源代码被包含在多个文件中时，这种差别才会显现出来。这个主题将在第 21 天的课程中介绍。

12.3.4 寄存器变量

关键字 register 用于建议编译器将一个动态局部变量存储在处理器的寄存器中，而不是常规内存中。处理器的寄存器是什么呢？使用它有何优点呢？

计算机的中央处理器（CPU）包含一些被称为寄存器的数据存储单元。实际的数据运算（如加和减）是在 CPU 的寄存器中进行的。要操纵数据，CPU 必须将数据从内存移到寄存器中，执行操纵，然后再将结果返回到内存中。在寄存器和内存之间移动数据需要一定的时间。如果将变量保留在内存中，则操纵变量的速度要快得多。

通过在动态变量的定义中使用关键字 register，可以请求编译器将该变量存储在寄存器中。请看下面的范例：

```
void func1(void)
{
    register int x;
    /* Additional code goes here */
}
```

这里说的是“请求”，而不是“命令”。根据程序的需求，变量可能没有寄存器可用。在这种情况下，编译器将它视为一个常规的动态变量。换句话说，register 关键字只是建议，而不是命令。对于函数频繁使用的变量（如循环中的计数器），寄存器存储类型能带来极大的好处。

关键字 register 只能用于简单的数值变量，而不能用于数组和结构。另外，它也不能用于静态变量和外部变量。您不能定义指向寄存器变量的指针。

应　该	不 应 该
一定要对局部变量进行初始化，否则您将不知道它包含的值是什么。	对于只有少数几个函数需要使用的变量，不要将它声明为全局的。将它作为参数传递给函数会更合适。
一定要对全局变量进行初始化，虽然默认情况下，它们将被初始化为 0。如果总是对变量进行初始化，则可以避免忘记初始化局部变量这样的情况发生。	对于非数值变量、数组和结构，不要将它们声明为寄存器变量。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

12.4　局部变量和 main()函数

前面对局部变量的所有讨论对 main()函数以及所有其他的函数都适用。严格地说，main()函数和其他函数没有什么两样。操作系统执行程序时，将首先调用 main()函数；程序结束后，main()函数将控制权交还给操作系统。

这意味着程序开始执行后，main()函数中定义的局部变量便被创建，直到程序结束时，它们才被释放。对于 main()函数而言，静态局部变量的值在两次调用函数之间保持不变是毫无意义的，因为程序结束后，变量就不存在了。因此，在 main()函数中，静态局部变量和动态局部变量之间没有任何差别。您可以在 main()中定义静态局部变量，但并不会有实际效果。

应　该	不应该
请记住，在大多数方面，main()函数与其他函数类似。	不要在 main()函数中声明静态局部变量，因为这样做毫无意义。

12.5　应使用哪种存储类型

在决定为变量使用哪种存储类型时，表 12.1 可能会对您有所帮助。该表对 C 语言中的 5 种存储类型进行了总结。

表 12.1　　C 语言中的 5 种存储类型

存储类型	关 键 字	生命周期	定义位置	作 用 域
动态变量	无	临时	函数内	局部
静态变量	static	临时	函数内	局部
寄存器变量	register	临时	函数内	局部
外部变量	无[2]	永久	函数外	全局（所有文件）
静态外部变量	static	永久	函数外	全局（一个文件）

[1] 关键字 auto 是可选的；

[2] 在函数内，使用关键字 extern 来声明一个已经在其他地方定义过的静态外部变量。

在决定使用何种存储类型时，应尽可能使用动态存储类型，只在必要时才使用其他存储类型。下面是一些指导原则：

- 首先考虑动态局部存储类型；
- 如果变量（如循环计数器）将被频繁使用，则在其定义中加上关键字 register；
- 在除 main()函数之外的函数中，如果变量的值需要在两次调用函数之间保持不变，则将它声明为静态的；
- 对于程序中的大部分或全部函数都需要使用的变量，应将其声明为外部变量。

12.6　局部变量和代码块

至此，今天的课程只讨论了函数中的局部变量。大部分局部变量属于这种情况，但您也可以在程序块（用花括号括起的代码段）中声明局部变量。在代码块中声明变量时，变量声明必须位于代码块的开始位置。程序清单 12.5 是一个这样的例子。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 12.5　　block.c：在程序代码块中定义局部变量

```
1:  /* Demonstrates local variables within blocks. */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:      /* Define a variable local to main(). */
8:
9:      int count = 0;
10:
11:     printf("\nOutside the block, count = %d", count);
12:
13:     /* Start a block. */
14:     {
15:       /* Define a variable local to the block. */
16:
17:       int count = 999;
18:       printf("\nWithin the block, count = %d", count);
19:     }
20:
21:     printf("\nOutside the block again, count = %d\n", count);
22:     return 0;
23: }
```

该程序的输出如下：

```
Outside the block, count = 0
Within the block, count = 999
Outside the block again, count = 0
```

从该程序可知，代码块中定义的 count 与代码块外定义的 count 是彼此独立的。第 9 行定义了一个名为 count 的 int 变量，并将其初始化为 0。由于这个变量是在 main()函数的开始位置定义的，因此它在整个 main()函数中都可用。第 11 行打印变量 count 的值，以说明它被初始化为 0。第 14～19 行是一个代码块，其中第 17 行定义了另一个名为 count 的 int 变量，并将其值初始化为 999。第 18 行打印代码块内声明的 count 变量的值（999）。由于代码块到 19 行便结束了，因此 21 行的打印语句打印的是第 9 行声明的 count 变量的值。

在 C 语言编程中，这种局部变量不常用，您也许永远不会遇到需要定义这种变量的情况。这种变量最常见的用途是用于隔离程序中的问题。可以使用花括号来暂时将一段代码隔离，并在其中创建局部变量来帮助查找问题。这种变量的另一个优点是，声明/初始化变量的代码和使用变量的代码很近，这有助于理解程序。

应　读	不应该
可（暂时）在代码块的开始位置使用变量来查找问题。	不要将变量定义放在除函数开头和代码块开头之外的其他位置。 不要在代码块的开头定义变量，除非这样做可使程序更清晰。

12.7　总　结

今天的课程介绍了变量存储类型的作用域和生命周期。每个变量（无论是简单变量、数组、结构，还是其他）都有特定的存储类型，它决定了变量的作用域（在程序的哪些地方可见）和生命周期（变量在内存中的存活时间）。

对于结构化编程来说，正确地使用存储类型至关重要。通过在函数中声明大多数变量，可以提高函数间

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

的独立性。除非有特殊的原因要求将变量声明为外部或静态的，否则应将变量声明为动态局部的。

12.8　问与答

问：既然全局变量在程序的任何地方都可用，为何不将所有的变量都声明为全局的呢？

答：随着程序越来越大，它包含的变量将越来越多。全局变量在程序运行期间一直保留在内存中，而动态局部变量只在它所在的函数执行期间保留在内存中。因此使用局部变量可以减少占有的内存量；更重要的是，可以降低程序不同部分的相互影响，从而减少程序 bug，并遵循结构化编程的原则。

问：第 11 天的课程指出，作用域对结构实例有影响，但不会影响结构名和结构体。原因何在呢？

答：定义结构时，如果没有同时声明实例，则创建了一个模板（或定义），但没有声明任何变量。仅当您创建结构的实例后，才声明了需要占用内存、并有作用域的变量。因此，可以在函数外面定义结构，而不会对内存有任何影响。很多程序员将常用的结构名和结构体放在头文件中，然后在需要创建这些结构的实例时，将该头文件包含进来（有关头文件的内容，将在第 21 天的课程中介绍）。

问：计算机是如何区分同名的全局变量和局部变量的？

答：这个问题超出了本书的范围。您需要知道的是，当局部变量的名称与全局变量名称相同时，在局部变量的作用域内（该局部变量所在的函数中），程序将暂时忽略全局变量，直到进入局部变量的作用域之外。

问：可以声明名称相同、但类型不同的局部变量和全局变量吗？

答：可以。名称相同的局部变量和全局变量是两个完全不同的变量。这意味着您可以将局部变量声明为任何数据类型。然而，在声明同名的局部变量和全局变量时要小心。有些程序员在命名全局变量时，总是以“g”打头（例如，命名为 gCount，而不是 Count）。这使得源代码中哪些是全局变量，哪些是局部变量一目了然。

12.9　作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

12.9.1　小测验

1. 作用域指的是什么？
2. 局部变量和全局变量之间最重要的差别是什么？
3. 变量的定义位置对其存储类型有何影响？
4. 定义局部变量时，影响变量的生命局期的选项有哪两个？
5. 定义动态和静态局部变量时，可以对它们进行初始化。初始化工作是在什么时候完成的？
6. 判断题：寄存器变量总是被保存在寄存器中？
7. 未被初始化的全局变量的值是什么？
8. 未被初始化的局部变量的值是什么？
9. 在程序清单 12.5 中，如果删除第 9 和 11 行，则第 21 行打印的内容是什么？请首先想一想，然后运行程序，看看结果是什么。
10. 如果函数需要在两次调用之间记住一个 int 类型的局部变量的值，应如何声明该局部变量？
11. 关键字 extern 有何用途？
12. 关键字 static 有何用途？

12.9.2　练习

1. 声明一个将被存储在 CPU 的寄存器中的变量。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

2. 修改程序清单 12.2，以避免其中的错误。修改时不要使用外部变量。

3. 编写一个程序，该程序声明并初始化一个 int 类型的、名为 var 的全局变量，然后在一个函数（不是 main()函数）中打印该变量的值。需要将 var 作为参数传递给该函数吗？

4. 修改练习 3 中编写的程序。不将 var 声明为全局变量，而在 main()函数中将其声明为局部变量。程序仍将使用一个函数来打印 var。在这种情况下，需要将 var 作为参数传递给该函数吗？

5. 程序中可以包含同名的全局变量和局部变量吗？编写一个程序来证明您的答案，该程序使用一个全局变量和一个局部变量，并且这两个变量的名称相同。

6. 排错：您能找出下述代码中的问题吗？（提示：问题与变量的声明位置有关。）

```
void a_sample_function( void )
{
    int ctr1;

    for ( ctr1 = 0; ctr1 < 25; ctr1++ )
      printf( "*" );

    puts( "\nThis is a sample function" );
    {
      char star = '*';
      puts( "\nIt has a problem\n" );
      for ( int ctr2 = 0; ctr2 < 25; ctr2++ )
      {
        printf( "%c", star);
      }
    }
}
```

7. 排错：下述代码有何错误？

```
/*Count the number of even numbers betweeno and 100.*/
#include <stdio.h>

int main( void )
{
     int x = 1;
     static int tally = 0;

     for (x = 0; x < 101; x++)
     {
         if (x % 2 == 0)  /*if x is even...*/
         tally++;         /*add 1 to tally.*/

     }

     printf("There are %d even numbers.\n", tally);
     return 0;
}
```

8. 排错：下述程序有何问题？

```
#include <stdio.h>

void print_function( char star );

int ctr;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
int main( void )
{
      char star;

      print_function( star );
      return 0;
}

void print_function( char star )
{
      char dash;

      for ( ctr = 0; ctr < 25; ctr++ )
      {
          printf( "%c%c", star, dash );
      }
}
```

9. 下面的程序打印什么内容？不要运行该程序，而应通过阅读代码来回答这个问题。

```
#include <stdio.h>
void print_letter2(void);          /* function prototype */

int ctr;
char letter1 = 'X';
char letter2 = '=';

int main( void )
{
      for( ctr = 0; ctr < 10; ctr++ )
      {
         printf( "%c", letter1 );
         print_letter2();
      }
      return 0;
}

void print_letter2(void)
{
      for( ctr = 0; ctr < 2; ctr++ )
       printf( "%c", letter2 );
}
```

10. 排错：上述程序能够运行吗？如果不能，问题何在？请修改它，使之能正确运行。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

TYPE & RUN 4　机密消息

这是第 4 个 Type & Run。别忘了，Type & Run 中的程序清单旨在提供一些功能比课程中的程序强些的程序。该程序清单包含很多前面介绍过的内容，同时还有一些以后的课程将介绍的内容，这包括使用磁盘文件（将在第 16 天的课程中介绍）。

下面的程序使您能够对机密消息进行加密和解密。运行该程序时，需要提供两个命令行参数：

```
coder filename action
```

其中，filename 是您创建的用来存储机密消息的文件或包含要解密的机密消息的文件；action 可以是 D 或 C，前者对机密消息进行解密，而后者对机密消息进行加密。如果您运行该程序时没有提供任何参数，程序将告诉您如何输入正确的参数。

由于该程序执行加密和解密，因此您可以将一个拷贝提供给朋友或同事，然后将加密后的机密消息发送给他们。他们可以使用该程序对消息进行解密，而没有该程序的人将无法读懂文件中的消息。

程序清单 R&T 4　　**Coder.c**

```
/* Program:  Coder.c
 * Usage:    Coder  [filename] [action]
 *            where filename = filename for/with coded data
 *            where action = D for decode anything else for
 *                           coding
 *-------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int encode_character( int ch, int val );
int decode_character( int ch, int val );

int main( int argc, char *argv[])
{
    FILE *fh;              /* file handle  */
    int rv = 1;            /* return value */
    int ch = 0;            /* variable to hold a character */
    unsigned int ctr = 0;   /* counter */
    int val = 5;           /* value to code with */
    char buffer[257];       /* buffer */

    if( argc != 3 )
    {
      printf("\nError:  Wrong number of parameters..." );
      printf("\n\nUsage:\n   %s filename action", argv[0]);
```

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
28:       printf("\n\n   Where:");
29:       printf("\n       filename - name of file to code or decode");
30:       printf("\n       action   = D for decode or C for encode\n\n");
31:       rv = -1;       /* set return error value */
32:    }
33:    else
34:    if(( argv[2][0] == 'D') || (argv [2][0] =='d'))/* to decode */
35:    {
36:       fh = fopen(argv[1], "r");   /* open the file   */
37:       if( fh <= 0 )               /* check for error */
38:       {
39:          printf( "\n\nError opening file..." );
40:          rv = -2;                 /* set return error value */
41:       }
42:       else
43:       {
44:          ch = getc( fh );     /* get a character */
45:          while( !feof( fh ) )  /* check for end of file */
46:          {
47:             ch = decode_character( ch, val );
48:             putchar(ch);  /* write the character to screen */
49:             ch = getc( fh);
50:          }
51:
52:          fclose(fh);
53:          printf( "\n\nFile decoded to screen.\n" );
54:       }
55:    }
56:    else  /* assume coding to file. */
57:    {
58:
59:       fh = fopen(argv[1], "w");
60:       if( fh <= 0 )
61:       {
62:          printf( "\n\nError creating file..." );
63:          rv = -3;  /* set return value */
64:       }
65:       else
66:       {
67:          printf("\n\nEnter text to be coded. ");
68:          printf("Enter a blank line to end.\n\n");
69:
70:          while( gets(buffer) != NULL )
71:          {
72:             if( buffer[0] == 0 )
73:                break;
74:
75:             for( ctr = 0; ctr < strlen(buffer); ctr++ )
76:             {
77:                ch = encode_character( buffer[ctr], val );
78:                ch = fputc(ch, fh);    /* write the character to file
*/
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
79:                 }
80:             }
81:             printf( "\n\nFile encoded to file.\n" );
82:             fclose(fh);
83:         }
84:
85:     }
86:     return (rv);
87: }
88:
89: int encode_character( int ch, int val )
90: {
91:    ch = ch + val;
92:    return (ch);
93: }
94:
95: int decode_character( int ch, int val )
96: {
97:     ch = ch - val;
98:     return (ch);
99: }
```

下面是一条加密后的消息：

```
Ymnx%nx%f%xjhwjy%rjxxflj&
```

将该消息解密后，结果如下：

```
This is a secret message!
```

该程序通过将字符加上或减去一个值，来对消息进行加密或解密。这很容易被破解。可以使用下面的代码替换第 91 和 97 行，以增加破解的难度：

```
ch = ch ^ val;
```

^是一个二目运算符，它通过位运算来修改字符。这样，可提高消息的机密度。

如果要将该程序提供给很多人，则可能需要填加第三个命令行参数。该参数用于设置 val 的值。变量 val 存储用于加密和解密的值。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第 13 天课程　高级程序流程控制

第 6 天的课程介绍了一些程序控制语句，它们控制程序中其他语句的执行。今天的课程将介绍有关程序控制的一些高级主题，包括 goto 语句以及使用循环能够完成的一些更有趣的工作。今天介绍以下内容：

- 如何使用 break 和 continue 语句；
- 何为死循环，为何使用这种循环？
- 何为 goto 语句，为何应避免使用它？
- 如何使用 switch 语句？
- 如何控制程序的退出？
- 如何在程序结束前自动执行函数？
- 如何在程序中执行系统命令？

13.1　提早结束循环

第 6 天的课程介绍了 for、while 和 do...while 循环是如何控制程序的执行的。这些循环根据程序中的条件执行代码块一次、多次，或一次也不执行。在这三种情况中，循环都是在满足特定的条件时结束。

然而，有时候您可能想进一步控制循环，语句 break 和 continue 提供了这样的控制功能。

13.1.1　break 语句

break 语句只能位于 for 循环、while 循环或 do...while 循环的循环体中（也可以位于 switch 语句中，有关这方面的内容，将在后面的课程中介绍）。遇到 break 语句后，程序将立刻退出循环。下面是一个这样的例子：

```
for ( count = 0; count < 10; count++ )
{
    if ( count == 5 )
    break;
}
```

如果没有 break 语句，该 for 循环将执行 10 次。但第六次迭代时，count 等于 5，因此执行 break 语句，导致 for 循环终止。然后，接着执行 for 循环中的右花括号后面的语句。执行嵌套 for 循环中的 break 语句时，将导致程序退出最内层的循环。

程序清单 13.1 演示了 break 语句的用法。

程序清单 13.1　　breaking.c：使用 break 语句

```
1:  /* Demonstrates the break statement. */
2:
3:  #include <stdio.h>
4:
5:  char s[] = "This is a test string. It contains two sentences.";
```

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
6:
7:  int main( void )
8:  {
9:     int count;
10:
11:     printf("\nOriginal string: %s", s);
12:
13:     for (count = 0; s[count]!='\0'; count++)
14:     {
15:        if (s[count] == '.')
16:        {
17:           s[count+1] = '\0';
18:           break;
19:        }
20:     }
21:     printf("\nModified string: %s\n", s);
22:
23:     return 0;
24: }
```

该程序的输出如下：

```
Original string: This is a test string. It contains two sentences.
Modified string: This is a test string.
```

分析：该程序取出字符串中的第一个句子。它逐个字符地搜索字符串中的第一个句点（该句点表明第一个句子到此结束）。这是通过一个for循环（第13～20行）来完成的。For循环从第13行开始，该行将count的值递增，以遍历字符串中的字符。第15行检查当前的字符是否为句点。如果是，则在句点后插入一个空字符（第17行）。这实际上是将字符串截断。截断字符串后，不需要再继续执行循环，因此第18行使用一条break语句来终止循环，以便接着执行循环后的第一行代码（第21行）。如果没有找到句点，则不对字符串做任何修改。

一个循环中可以包含多条break语句，但只有首次被执行的语句（如果有的话）有效。如果没有任何break语句被执行，则循环将（根据测试条件）正常终止。图13.1说明了break语句的工作原理。

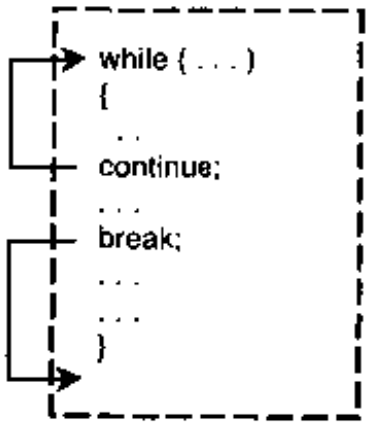


图13.1　break语句和continue语句的工作原理

break语句的语法如下：

```
break;
```

break语句用于循环或switch语句中，它导致程序立刻终止当前的循环（for、while或do...while）或switch语句。这样，余下的循环迭代不再执行，而接着执行循环或switch语句后面的第一条语句。

下面是一个使用break语句的范例：

```
int x;
printf ( "Counting from 1 to 10\n" );
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
/* Having no condition in the for loop will cause it to loop forever */

for( x = 1; ; x++ )
{
   if( x >= 10 )    /* This checks for the value of 10 */
      break;        /* This ends the loop */
   printf( "\n%d", x );
}
```

13.1.2　continue 语句

与 break 语句一样，continue 语句也只能位于 for、while 和 do...while 循环的循环体内。执行 continue 语句后，立刻执行循环的下一次迭代，continue 语句到循环体结尾之间的语句不会被执行。图 13.1 说明了 continue 语句的工作原理，注意它与 break 语句之间的差别。

程序清单 13.2 使用了 continue 语句。该程序从键盘接受一行输入，然后删除其中小写的元音字符，并显示余下的内容。

程序清单 13.2　　cont.c：使用 continue 语句

```
1:  /* Demonstrates the continue statement. */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:     /* Declare a buffer for input and a counter variable. */
8:
9:     char buffer[81];
10:    int ctr;
11:
12:    /* Input a line of text. */
13:
14:    puts("Enter a line of text:");
15:    gets(buffer);
16:
17:    /* Go through the string, displaying only those */
18:    /* characters that are not lowercase vowels. */
19:
20:    for (ctr = 0;buffer[ctr] != '\0'; ctr++)
21:    {
22:
23:       /* If the character is a lowercase vowel, loop back */
24:       /* without displaying it. */
25:
26:       if (buffer[ctr] == 'a' || buffer[ctr] == 'e'
27:          || buffer[ctr] == 'i' || buffer[ctr] == 'o'
28:          || buffer[ctr] == 'u')
29:            continue;
30:
31:       /* If not a vowel, display it. */
32:
33:       putchar(buffer[ctr]);
34:    }
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
35:     return 0;
36: }
```

该程序的运行情况如下：

```
Enter a line of text:
This is a line of text
Ths s  ln f txt
```

分析：虽然该程序不是很实用，但它确实高效地使用了 continue 语句。第 9 和 10 行声明了变量。数组 buffer[]用于存储用户输入（第 15 行）的字符串。另一个变量 ctr 用于在第 20～34 行的 for 循环中来遍历数组 buffer[]。在 for 循环中，第 26～28 行的 if 语句检查字符是否为小写的元音字符。如果是，则执行 continue 语句，程序转到第 20 行；如果不是，则执行第 33 行。第 33 行使用了一个新的库函数 putchar()，该函数将一个字符显示到屏幕上。

continue 语句的语法如下：

```
continue;
```

continue 语句用于循环中。它使程序结束当前的循环迭代，并立刻进行下一次迭代。

下面是一个使用 continue 语句的范例：

```
int x;
printf("Printing only the even numbers from 1 to 10\n");
for( x = 1; x <= 10; x++ )
{
   if( x % 2 != 0 )    /* See if the number is NOT even */
          continue;      /* Get next instance x */
   printf( "\n%d", x );
}
```

13.2 goto 语句

goto 语句是一种无条件转移（或分支）语句。程序执行到 goto 语句后，立刻跳转到它指定的位置。这种语句是无条件的，因为不管情况如何，程序执行到这里总是会跳转（而不像 if 语句那样）。

Goto 语句的跳转目标是由位于行首的文本标签和分号标识的。目标标签可以单独占一行，也可位于 C 语句行的行首。程序中的每个目标标签都必须是独一无二的。

goto 语句和它跳转到的语句必须位于同一个函数中，但可以位于不同的代码块中。程序清单 13.3 是一个使用 goto 语句的简单程序。

程序清单 13.3　　gotolt.c：使用 goto 语句

```
1:  /* Demonstrates the goto statement */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:      int n;
8:
9:  start:
10:
11:     puts("Enter a number between 0 and 10: ");
12:     scanf("%d", &n);
13:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
14:     if (n < 0 ||n > 10 )
15:         goto start;
16:     else if (n == 0)
17:         goto location0;
18:     else if (n == 1)
19:         goto location1;
20:     else
21:         goto location2;
22:
23: location0:
24:     puts("You entered 0.\n");
25:     goto end;
26:
27: location1:
28:     puts("You entered 1.\n");
29:     goto end;
30:
31: location2:
32:     puts("You entered something between 2 and 10.\n");
33:
34: end:
35:     return 0;
36: }
```

该程序的运行情况如下：

```
Enter a number between 0 and 10:
1
You entered 1.
Enter a number between 0 and 10:
9
You entered something between 2 and 10.
```

分析：该程序从键盘读取一个 0～10 之间的数字。如果用户输入的数字不在 0～10 之间，则程序使用一条 goto 语句（第 15 行）跳转到 start 处（第 9 行）。否则检查该数字是否为 0（第 16 行），如果是，则执行第 17 行的 goto 语句，跳转到 location0 处（第 23 行），然后打印一条消息（第 24 行），并执行另一条 goto 语句。第 25 行的 goto 语句跳转到程序末尾的 end 处。对于 1 以及 2～10 之间的数字，程序执行类似的逻辑。

goto 语句的跳转目标可以在前面，也可以在后面。正如前面指出的，唯一的限制是，goto 语句及其跳转目标必须位于同一个函数中，但可以位于不同的代码块中。可以使用 goto 语句跳转到循环外或从循环外跳转到循环内，但决不要这样做。事实上，我们强烈建议您决不要在程序中使用 goto 语句。原因有两点：

- 您不需要使用这种语句。没有什么编程任务必须使用 goto 语句才能完成，您总是可以使用其他的分支语句来完成。
- 这种语句很危险。对于有些编程问题来说，goto 语句好像是一种理想的解决方案，但它不遵守运行规程。使用 goto 语句来转移程序流程时，无法知道以前的执行位置，因此导致程序的执行情况杂乱无章。这种编程方式被称为意大利面条式代码。

有些细心的程序员能够使用 goto 语句编写出不错的程序。也存在这样的情况，即大量地使用 goto 语句是完成编程任务的最简单的解决方案；但这决不是唯一的解决方案。如果您不听劝告，一定要使用 goto 语句，那么使用时一定要小心。

应　该	不 应 该
在可能的情况下，不要使用 goto 语句（这总是可能的）。	不要将 break 语句和 continue 语句混淆，前者结束循环，而后者结束循环的当前迭代，直接进入下一次迭代。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

goto 语句的语法如下：

```
goto location;
```

其中 location 是一条标签语句，标识了要跳转到的位置。标签语句由标识符、冒号和一条语句（可选）组成：

```
location: a C statement;
```

也可以让标签本身单独占一行。这样做时，有些程序员喜欢在它后面加上一条空语句（一个分号），虽然不一定非要这样：

```
location: ;
```

13.3 死循环

何为死循环，为何您想在程序中使用这种循环呢？死循环指的是如果没有外力的干预，将永远执行下去的循环。这可以是 for 循环、while 循环，也可以是 do...while 循环。例如，下面就是一个死循环：

```
while (1)
{
    /* additional code goes here */
}
```

while 测试的条件为常量 1，它总为真，而且程序无法改变该条件。由于 1 永远不会改变自身，因此循环将不会终止。

前一节介绍过，可以使用 break 语句来退出循环。如果没有 break 语句，则死循环将毫无用处；但通过 break 语句，死循环便可以发挥其作用。

您也可以编写 for 死循环，如下所示：

```
for (;;)
{
    /* additional code goes here */
}
```

你还可以编写 do...while 死循环，如下所示：

```
do
{
    /* additional code goes here */
} while (1);
```

这三类循环的原理是相同的。在后面的范例中，将使用 while 循环。

当需要测试很多条件，以确定是否需要结束循环时，可以使用死循环。要将所有测试条件放在 while 语句的圆括号中可能很困难，而在循环体中对各个条件进行测试，并必须执行 break 语句才能退出循环则可能容易得多。通常，如果有其他办法时，应避免使用死循环。

也可以使用死循环创建一个菜单系统，以引导程序的运行。您可能还记得，在第 5 天的课程中，有一个程序的 main()函数充当了“交通警察”的角色，它引导程序执行各种函数，程序的实际工作是由这些函数完成的。这通常是使用某种菜单来完成的：程序显示一个选项列表供用户选择，其中的一个选项用于结束程序。用户做出选择后，程序通过一条决策语句来执行相应的代码。

程序清单 13.4 演示了一个菜单系统。

程序清单 13.4　　menu.c：使用死循环实现菜单系统

```
1:  /* Demonstrates using an infinite loop to implement */
2:  /* a menu system. */
3:  #include <stdio.h>
4:  #define DELAY  150000      /* Used in delay loop. */
5:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
6:  int menu(void);
7:  void delay(void);
8:
9:  int main( void )
10: {
11:    int choice;
12:
13:    while (1)
14:    {
15:
16:     /* Get the user's selection. */
17;
18:     choice = menu();
19:
20:     /* Branch based on the input. */
21:
22;    if (choice == 1)
23:    {
24:       puts("\nExecuting task A.");
25:       delay();
26:    }
27:    else if (choice == 2)
28:        {
29:           puts("\nExecuting task B.");
30:           delay();
31:        }
32;    else if (choice == 3)
33:        {
34:           puts("\nExecuting task C.");
35:           delay();
36:        }
37:    else if (chcice == 4)
38:        {
39:           puts("\nExecuting task D.");
40:           delay();
41:        }
42:     else if (choice == 5)      /* Exit program. */
43:        {
44:           puts("\nExiting program now...\n");
45:           delay();
46:           break;
47:        }
48:        else
49:        {
50:           puts("\nInvalid choice, try again.");
51:           delay();
52:        }
53:    }
54:    return 0;
55: }
56:
57: /* Displays a menu and inputs user's selection. */
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
58: int menu(void)
59: {
60:     int reply;
61:
62:     puts("\nEnter 1 for task A.");
63:     puts("Enter 2 for task B.");
64:     puts("Enter 3 for task C.");
65:     puts("Enter 4 for task D.");
66:     puts("Enter 5 to exit program.");
67:
68:     scanf("%d", &reply);
69:
70:     return reply;
71: }
72:
73: void delay( void )
74: {
75:     long x;
76:     for ( x = 0; x < DELAY; x++ )
77:         ;
78: }
```

该程序的运行情况如下：

```
Enter 1 for task A.
Enter 2 for task B.
Enter 3 for task C.
Enter 4 for task D.
Enter 5 to exit program.
1

Executing task A.

Enter 1 for task A.
Enter 2 for task B.
Enter 3 for task C.
Enter 4 for task D.
Enter 5 to exit program.
6
Invalid choice, try again.

Enter 1 for task A.
Enter 2 for task B.
Enter 3 for task C.
Enter 4 for task D.
Enter 5 to exit program.
5

Exiting program now...
```



警告：程序清单13.4没有包含任何检查错误的代码。如果用户输入的不是数字，则运行结果将是不可预测的。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

分析：程序清单 13.4 的第 18 行调用了一个名为 menu()的函数，该函数是在第 58～71 行定义的。menu()函数在屏幕上显示一个菜单、读取用户的输入，并将输入返回给主程序。main()函数则使用一系列的嵌套 if 语句来检查返回的值，并执行相应的代码。该程序的唯一功能是将消息显示到屏幕上。在实际的程序中，代码将调用各种函数来完成用户选择的任务。

该程序还使用了另一个函数——delay()。该函数是在第 73～78 行定义的，它完成的工作不多。简单地说，第 76 行的 for 语句执行循环，但循环时没有执行任何操作（第 77 行）。该语句循环 DELAY 次，这是一种让程序暂停一段时间的有效的方法。如果暂停的时间过长或过短，则可以相应地调整 DELAY 的值。

很多编译器包含一个类似于 delay()的函数。Borland 和 Symantec 的编译器包含一个名为 sleep()的函数，该函数让程序暂停传递给它的参数指定的秒数。如果使用的是 Symantec 的编译器，则要使用 sleep()函数，程序必须包含头文件 time.h；如果使用的是 Borland 的编译器，则必须包含头文件 dos.h。如果您使用的是上述两种编译器或其他支持 sleep()的编译器，则可以使用 sleep 函数代替 delay()。



> 警告：还有比程序清单 13.4 更好的方法来使程序暂停。如果您选择使用一个像前面介绍的 sleep()这样的函数，则一定要小心。sleep()函数不是 ANSI-兼容的，这意味着，使用其他编译器时，该函数不管用；同时它也不一定适用于所有平台。

13.4　switch 语句

在 C 语言中，最灵活的程序控制语句是 switch 语句，它使您能够根据表达式的值（多于两个）来执行不同的语句。前面介绍的控制语句，如 if 语句，只能对可能的取值为两个（真或假）的表达式进行判断。为根据两个以上的值来控制程序的流程，可以使用多个嵌套的 if 语句，如程序清单 13.4 所示。而使用了 switch 语句，就无需使用嵌套语句。

switch 语句的通用格式如下：

```
switch (expression)
{
    case  template_1: statement(s);
    case  template_2: statement(s);
    ...
    case  template_n: statement(s);
    default: statement(s);
}
```

其中，expression 可以是任何结果为整数（类型为 long、int 或 char）的表达式。switch 语句计算 expression 的值，将其同 case 标签后的模板进行比较。然后发生下面的事件之一：

- 如果找到与 expression 匹配的模板，则执行该模板后面的语句；
- 如果没有找到任何匹配的模板，则执行（可选的）default 标签后面的语句；
- 如果没有找到任何匹配的模板，且没有 default 标签，则执行 switch 语句中的右花括号后面的语句。

程序清单 13.5 演示了 switch 语句的用法，该程序根据用户的输入显示一条消息。

程序清单 13.5　　switch.c：使用 switch 语句

```
1:  /* Demonstrates the switch statement. */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
7:     int reply;
8:
9:     puts("Enter a number between 1 and 5:");
10:    scanf("%d", &reply);
11:
12:    switch (reply)
13:    {
14:       case 1:
15:          puts("You entered 1.");
16:       case 2:
17:          puts("You entered 2.");
18:       case 3:
19:          puts("You entered 3.");
20:       case 4:
21:          puts("You entered 4.");
22:       case 5:
23:          puts("You entered 5.");
24:       default:
25:          puts("Out of range, try again.");
26:    }
27:
28:    return 0;
29: }
```

该程序的运行情况如下：

```
Enter a number between 1 and 5:
2
You entered 2.
You entered 3.
You entered 4.
You entered 5.
Out of range, try again.
```

分析：显然结果是不正确的。看起来好像switch语句发现第一个模板匹配，然后执行后面的所有代码，而不仅仅是与此模板相关的语句。情况确实如此，然而，switch语句确实是这样工作的。实际上，它跳转到匹配的模板处。为确保只执行与匹配的模板相关的语句，应在必要的地方包含一条break语句。程序清单13.6是添加break语句后的程序，现在它能够正确运行。

程序清单13.6　switch2.c：正确地使用switch——在必要的地方加上break语句

```
1:  /* Demcnstrates the switch statement correctly. */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:     int reply;
8:
9:     puts("\nEnter a number between 1 and 5:");
10:    scanf("%d", &reply);
11:
12:    switch (reply)
13:    {
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
14:     case 0:
15:        break;
16      case 1:
17:       {
18:        puts("You entered 1.\n");
19:        break;
20:       }
21:     case 2:
22:       {
23:        puts("You entered 2.\n");
24:        break;
25:       }
26:     case 3:
27:       {
28:        puts("You entered 3.\n");
29:        break;
30:       }
31:     case 4:
32:       {
33:        puts("You entered 4.\n");
34:        break;
35:       }
36:     case 5:
37:       {
38:        puts("You entered 5.\n");
39:        break;
40:       }
41:     default:
42:       {
43:        puts("Out of range, try again.\n");
44:       }
45:   }            /* End of switch */
46:   return 0;
47: }
```

该程序的运行情况如下：

```
Enter a number between 1 and 5:
1
You entered 1.
Enter a number between 1 and 5:
6
Out of range, try again.
```

switch 语句常见的用途之一是用于实现程序清单 13.4 中那样的菜单。程序清单 13.7 使用 switch 语句（而不是 if 语句）来实现菜单。使用 switch 语句比使用 if 语句要好得多，而程序清单 13.4 使用的便是 if 语句。

程序清单 13.7　　menu2.c：使用 switch 语句实现菜单系统

```
1:  /* Demonstrates using an infinite loop and the switch */
2:  /* statement to implement a menu system. */
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
#define DELAY 150000

int menu(void);
void delay(void);

int main( void )
{
   int command = 0;
   command = menu();

   while (command != 5 )
   {
        /* Get user's selection and branch based on the input.*/

        switch(command)
        {
        case 1:
           {
            puts("\nExecuting task A.");
            delay();
            break;
           }
        case 2:
           {
            puts("\nExecuting task B.");
            delay();
            break;
           }
        case 3:
           {
            puts("\nExecuting task C.");
            delay();
            break;
           }
        case 4:
           {
              puts("\nExecuting task D.");
              delay();
              break;
          . }
        case 5:    /* Exit program. */
           {
            puts("\nExiting program now...\n");
           }
        default:
           {
            puts("\nInvalid choice, try again.");
           }
      }  /* End of switch */
      command = menu();
   }      /* End of while  */
   return 0;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
58: }
59:
60: /* Displays a menu and inputs user's selection.*/
61: int menu(void)
62: {
63:     int reply;
64:
65:     puts("\nEnter 1 for task A.");
66:     puts("Enter 2 for task B.");
67:     puts("Enter 3 for task C.");
68:     puts("Enter 4 for task D.");
69:     puts("Enter 5 to exit program.");
70:
71:     scanf("%d", &reply);
72:
73:     return reply;
74: }
75:
76: void delay( void )
77: {
78:     long x;
79:     for( x = 0; x < DELAY; x++ )
80:         ;
81: }
```

该程序的运行情况如下：

```
Enter 1 for task A.
Enter 2 for task B.
Enter 3 for task C.
Enter 4 for task D.
Enter 5 to exit program.
1

Executing task A.

Enter 1 for task A.
Enter 2 for task B.
Enter 3 for task C.
Enter 4 for task D.
Enter 5 to exit program.
6

Invalid choice, try again.

Enter 1 for task A.
Enter 2 for task B.
Enter 3 for task C.
Enter 4 for task D.
Enter 5 to exit program.
5

Exiting program now...
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

分析：该程序使用一条 switch 语句，根据用户选择的菜单项执行相应的操作。第 14 行调用 menu()函数，并获得用户选择的值。如果值不为 5，则执行 while 循环。该 while 语句主要由一个 switch 语句组成，后者根据用户选择的菜单项执行不同的代码。执行相应的代码后，系统再次显示菜单并获得用户做出的选择（第 55 行）。

有时候，让程序依次执行一系列 case 很有用。例如，假设您希望表达式的值为多个值中的任何一个时，都执行相同的语句块，则只需省略 break 语句，并将所有的 case 模板都放在相应语句块的前面。如果测试表达式与其中的任何一个 cast 条件匹配，则沿 case 语句不断执行下去，最后执行指定的语句块。程序清单 13.8 演示了这种用法。

程序清单 13.8　　fallthru.c：switch 语句的另一种用法

```
1:  /* Another use of the switch statement. */
2:
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:
6:  int main( void )
7:  {
8:     int reply;
9:
10:     while (1)
11:     {
12:        puts("\nEnter a value between 1 and 10, 0 to exit: ");
13:        scanf("%d", &reply);
14:
15:        switch (reply)
16:        {
17:           case 0:
18:              exit(0);
19:           case 1:
20:           case 2:
21:           case 3:
22:           case 4:
23:           case 5:
24:              {
25:                puts("You entered 5 or below.\n");
26:                break;
27:              }
28:           case 6:
29:           case 7:
30:           case 8:
31:           case 9:
32:           case 10:
33:              {
34:                puts("You entered 6 or higher.\n");
35:                break;
36:              }
37:           default:
38:                puts("Between 1 and 10, please!\n");
39:        } /* end of switch */
40:     }    /*end of while */
41:     return 0;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
43: }
```

该程序的运行情况如下：

```
Enter a value between 1 and 10, 0 to exit:
11
Between 1 and 10, please!

Enter a value between 1 and 10, 0 to exit:
1
You entered 5 or less.
Enter a value between 1 and 10, 0 to exit:
5
You entered 6 or more.

Enter a value between 1 and 10, 0 to exit:
0
```

分析：该程序从键盘读取一个值，然后指出这个值是小于等于 5、大于等于 6 或者不在 1～10 之间。如果用户输入的值为 0，则程序调用 exit()函数（第 18 行），从而结束程序。

您不能像程序清单 13.4 那样，在第 18 行使用一条 break 语句。break 语句只是跳出 switch 语句，而不能跳出 while 死循环。exit()函数终止程序，下一节将对该函数做更详细的介绍。

switch 语句的语法如下：

```
switch (expression)
{
        case template_1: statement(s);
        case template_2: statement(s);
        ...
        case template_n: statement(s);
        default: statement(s);
}
```

switch 语句让您能够根据一个表达式的值来执行多个不同的分支。其效率比多层嵌套的 if 语句高，也更容易理解。switch 语句计算表达式的值，然后执行与表达式的值匹配的模板所在的 case 语句。如果没有与表达式的值匹配的模板，则执行 default 语句。如果没有 default 语句，则执行 switch 语句后面的语句。

程序将沿 case 语句不断往下执行，直到遇到 break 语句，然后执行 switch 语句后面的语句。

下面是两个使用 switch 语句的范例：

范例 1：

```
switch( letter )
{
    case 'A':
    case 'a':
       printf( "You entered A" );
       break;
    case 'B':
    case 'b':
       printf( "You entered B");
       break;
    ...
    ...
    default:
      printf( "I don't have a case for %C", letter);
}
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

范例 2：

```
switch( number )
{
      case 0:    puts( "Your number is 0 or less.");
      case 1:    puts( "Your number is 1 or less.");
      case 2:    puts( "Your number is 2 or less.");
      case 3:    puts( "Your number is 3 or less.");
      ...
      ...
      case 99:   puts( "Your number is 99 or less.");
            break;
      default:   puts( "Your number is greater than 99.");
}
```

在第二个例子中，由于开始的 case 语句中没有 break 语句，因此将找到与 number 匹配的 case 语句，并执行后续的所有 case 语句中的 puts()语句，直到 case99 中的 break 语句。如果 number 的值为 3，则程序将指出您输入的值小于等于 3、小于等于 4、小于等于 5，直到小于等于 99。程序将不断地打印，直到到达 case99 中的 break 语句。

应　该	不应该
一定要在 switch 语句中使用 default 语句，即使您认为已涵盖了所有的情况； 如果需要检查同一变量的两种以上的值，一定要使用 switch 语句，而不要使用 if 语句； 一定要将 case 语句对齐，以提高程序的可读性。	别忘了必要时在 switch 语句中使用 break 语句。

13.5　退出程序

正常情况下，程序在执行到 main()函数的右花括号时结束；然而，您可以随时调用库函数 exit()来终止程序。也可以指定一个或多个在程序结束时自动执行的函数。

13.5.1　exit()函数

exit()函数终止程序的执行，并将控制权归还给操作系统。该函数接受一个 int 类型的参数，并将它传递给操作系统，指出程序是正常还是异常终止。exit()函数的语法如下：

```
exit(status);
```

如果 status 为 0，则表明程序被正常终止；如果为 1，则表明程序终止时发生了某种错误。该函数的返回值通常被忽略。在 DOS 操作系统中，可以使用一个 DOS 批文件和 if errorlevel 语句来检测返回的值。本书并非介绍 DOS 的，因此如果您需要使用程序的返回值，请参考 DOS 文档。如果您使用的操作系统不是 DOS，则应查看相应的文档来获悉如何使用程序返回的值。

要使用 exit()函数，程序必须包含头文件 stdlib.h。该文件还定义了两个符号常量，用作 exit()函数的参数：

```
#define EXIT_SUCCESS   0
#define EXIT_FAILURE   1
```

因此，要退出程序并返回 0，可调用 eixt(EXIT_SUCCESS)；要返回 1，请调用 exit(EXIT_FUILURE)。

应　该	不应该
在出现问题时，一定要使用 exit()函数退出程序； 一定要将有意义的值传递给 exit()函数。	

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

13.6　在程序中执行操作系统命令

C 语言的标准库中包含一个 system()函数，它让您能够在一个正在运行的程序中执行操作系统命令。这很有帮助，它让您无需退出程序就能读取磁盘的目录列表或对磁盘进行格式化。要使用 system()函数，程序必须包含头文件 stdlib.h。system()函数的使用格式如下：

```
system(command);
```

其中，参数 command 可以是一个字符串常量，也可以是一个指向字符串的指针。例如，在 DOS 下，要获得目录列表，可以这样编写代码：

```
system("dir");
```

也可以这样编写代码：

```
char *command = "dir";
system(command);
```

执行完操作系统命令后，接着执行 system()调用后面的语句。如果传递给 system()的命令不是有效的操作系统命令，则系统将显示错误消息 Bad command or file name，然后继续执行后面的代码。程序清单 13.9 演示了 system()函数的用法。

程序清单 13.9　　system.c：使用 system()函数执行系统命令

```
1:  /* Demonstrates the system() function. */
2:  #include <stdio.h>
3:  #include <stdlib.h>
4:
5:  int main( void )
6:  {
7:      /* Declare a buffer to hold input. */
8:
9:      char input[40];
10:
11:     while (1)
12:     {
13:         /* Get the user's command.*/
14:
15:         puts("\nInput the desired system command, blank to exit");
16:         gets(input);
17:
18:         /* Exit if a blank line was entered. */
19:
20:         if (input[0] == '\0')
21:             exit(0);
22:
23:         /* Execute the command. */
24:
25:         system(input);
26:     }
27:     return 0;
28: }
```

该程序的运行情况如下：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn


```
Input the desired system command, blank to exit
dir *.bak

Volume in drive E is BRAD_VOL_B
Directory of E:\BOOK\LISTINGS
LIST_414 BAK      1416 05-22-99   5:18p
1 file(s)      1416 bytes
240068096 bytes free

Input the desired DOS command, blank to exit
```



注意：dir *.bak 是一个 DOS 命令，它命令操作系统列出当前目录中扩展名为.BAK 的所有文件。该命令在 Microsoft Windows 下也管用。对于 UNIX 机器，相应的命令为 ls *.bak。如果您使用的是 System 7 或其他操作系统，则应使用相应的操作系统命令。

分析：程序清单 13.9 演示了 system()的用法。该程序使用了一个 while 循环（第 11～26 行）让用户执行操作系统命令。第 15 和 16 行提示用户输入要执行的操作系统命令。如果用户直接按下 Enter 键，程序将调用 exit()（第 20 和 21 行）来结束程序。第 25 行调用 system()函数，并将用户输入的命令作为参数。当然，在您的计算机上运行该程序时，输出肯定与上面不同。

可传递给 system()的命令不仅仅是简单的操作系统命令，如列出目录或格式化磁盘；还可以将可执行文件或批文件的名称传递给它，而程序仍正常运行。例如，如果您传递 LIST1308，则执行名为 LIST1308 的程序。执行完被调用的程序后，接着执行 system()调用后面的代码。

使用 system()时，唯一的限制是内存量。调用 system()时，原来的程序仍然位于计算机的 RAM 中，同时将把一个操作系统命令处理器的拷贝和运行的程序装载到内存中。仅当有充足的内存时，这才能正常进行；否则系统将显示一条错误消息。



注意：在本书附带光盘中的编译器 BloodShed Dev-C++中，当您新建一个源代码文件时，该文件将使用函数 system()。如果您选择菜单 File/New Source file，则新创建的源代码文件如下：

```
#include <iostream.h>
#include <stdlib.h>

int main()
{

        system("PAUSE");
        return 0;
}
```

system("PAUSE")使程序暂停，直到用户按下任意键后。这样，用户可以查看程序的输出；否则用户还未看清结果之前，显示输出的窗口就关闭了。

13.7 总 结

今天的课程介绍了有关程序控制的各种主题。您学习了 goto 语句，并知道为何应避免使用这种语句。您学习了 break 和 continue 语句，它们让您能够更好地控制循环的执行。可结合使用这些语句和死循环，来完成一些很有用的编程任务。另外，您还学习了如何使用 exit()来终止程序。最后，您学习了如何在程序中使用 system()函数来执行操作系统命令。

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

13.8　问与答

问：应使用 switch 语句还是嵌套 if 语句？

答：如果要检查的变量有两个以上的可能取值，则 switch 语句更合适。这样编码的可读性也更高。如果要检查的是真/假条件，则应使用 if 语句。

问：为何应避免使用 goto 语句？

答：当您第一次接触到 goto 语句时，很可能认为它很有用。实际上，goto 语句弊大于利。goto 语句是一个非结构化命令，它跳转到程序的另一个地方。很多调试器（帮助您查找程序问题的软件）无法正确地处理 goto 语句。另外，goto 语句还会导致意大利面条式代码——代码的逻辑不清晰。

问：为何不同编译器提供的函数不同？

答：今天的课程中指出，对于有些函数，并非所有的编译器或计算机操作系统中都提供了。例如，Borland 的编译器提供了 sleep()函数，但 Microsoft 的编译器没有提供。

虽然存在一些所有 ANSI 编译器都必须遵循的标准，但这些标准并没有禁止编译器厂商添加其他的功能。编译器厂商通过创建和包括新的函数来添加功能，每个编译器厂商都会添加很多他们认为对用户有帮助的函数。

问：C 语言是一种标准化语言吗？

答：事实上，C 语言是高度标准化的。美国国家标准化组织（ANSI）制定了 ANSI C 标准，该标准几乎对 C 语言的所有细节都做出了规定，包括提供的函数。有些编译器厂商在其编译器中添加了其他函数（ANSI 标准中没有的函数），旨在在竞争中胜人一筹。另外，有些编译器没有遵守 ANSI 标准。对于那些遵守 ANSI 标准的编译器而言，99%的程序语法和函数都是相同的。

问：使用 system()执行系统功能好吗？

答：对于执行诸如列出目录中的文件等任务而言，使用 system()函数是一种很容易的途径，但您必须小心使用该函数。大多数操作系统命令随操作系统而异，如果您使用 system()，则代码将可能无法移植。如果是运行另一个程序（不是操作系统命令），则不存在移植性方面的问题。

13.9　作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

13.9.1　小测验

1. 何时应使用 goto 语句？
2. break 语句与 continue 语句之间有何区别？
3. 何为死循环，如何编写这样的循环？
4. 哪两种情况将导致程序终止？
5. switch 语句可以检查哪两种类型的变量？
6. default 语句有何用途？
7. exit()函数有何用途？
8. system()函数有何用途？

13.9.2　练习

1. 编写一条语句，使程序进入循环的下一次迭代。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

2. 编写一条语句，使程序终止循环。

3. 编写一行代码，显示当前目录中的所有文件（DOS 操作系统）。

4. 排错：下面的代码有错误吗？

```
switch( answer )
{
        case 'Y':printf ("you answered yes");
              break;
        case 'N':printf("you answered no");
}
```

5. 排错：下面的代码有错误吗？

```
switch( choice )
{
    default:
        printf("You did not choose 1 or 2");
    case 1:
        printf("You answered 1");
        break;
    case 2:
        printf( 'You answered 2");
        break;
}
```

6. 使用 if 语句重新编写练习 5 中的代码。

7. 编写一个 do...while 死循环。

由于下述练习的可能答案有多个，因此附录 F 没有提供这些练习的答案。这些练习是“选做题”。

8. 选做题：编写一个类似于计算器的程序，该程序能够执行加、减、乘、除运算。

9. 选做题：编写一个程序，该程序提供一个包含 5 个选项的菜单。第 5 个菜单项是退出程序；而其他的菜单项则使用 system()函数来执行一个系统命令。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 14 天课程　操纵屏幕、打印机和键盘

几乎所有的程序都必须执行输入和输出，程序处理输入和输出的能力是衡量其实用性的晴雨表。前面介绍了如何执行一些基本的输出和输出，今天将介绍以下内容：

- C 语言如何使用流来进行输入和输出；
- 各种从键盘读取输入的方式；
- 在屏幕上显示文本和数值数据的方法；
- 如何将输出发送给打印机？
- 如何重定向输入和输出？

14.1　流和 C 语言

详细介绍程序的输入/输出之前，先介绍一下流。在 C 语言中，所有的输入/输出操作都是通过流来完成的，而不管输入来自何方，输出将去往何方。稍后您将知道，对程序员而言，这种处理所有输入和输出的标准方式有很大的优点。当然，这要求您必须理解流的概念及其工作原理。首先，需要了解术语输出和输入到底指的是什么。

14.1.1　何为程序的输入/输出

正如本书前面介绍的，C 程序执行时将数据存储在随机存取器（RAM）中。这些数据是以程序定义的变量、结构和数组等形式存在的。这些数据来自什么地方？程序如何处理它们呢？

- 数据来自程序的外面。从外面被移到 RAM（以便程序能够存取）的数据叫做输入。最常见的程序输入源是键盘和磁盘文件。
- 程序也可以将数据发送到外面的某个地方，这被称为输出。最常见的输出目的地是屏幕、打印机和磁盘文件。

输入源和输出目的地被统称为设备。键盘是设备，屏幕也是设备，等等。有些设备（键盘）只能用来输入，另一些设备（屏幕）只能用于输出，还有一些设备（磁盘文件）则可用来输入和输出。图 14.1 对此做了说明。

无论是什么设备，也不管它用于输入还是输出，C 语言都采用流的方式执行输入和输出操作。

14.1.2　什么是流

流是一个字符序列。更准确地说，是一个数据字节序列。流入程序的字节序列为输入流；从程序流出的字节序列是输出流。对于流，您无需关心它来自哪里或者去往何方。因此，流的主要优点是，输入/输出编程是独立于设备的。程序员无需针对每种设备（键盘、磁盘等）编写特殊的输入/输出函数。程序将输入/输出视为连续的字节流，而不管输入来自哪里，输出去往何方。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

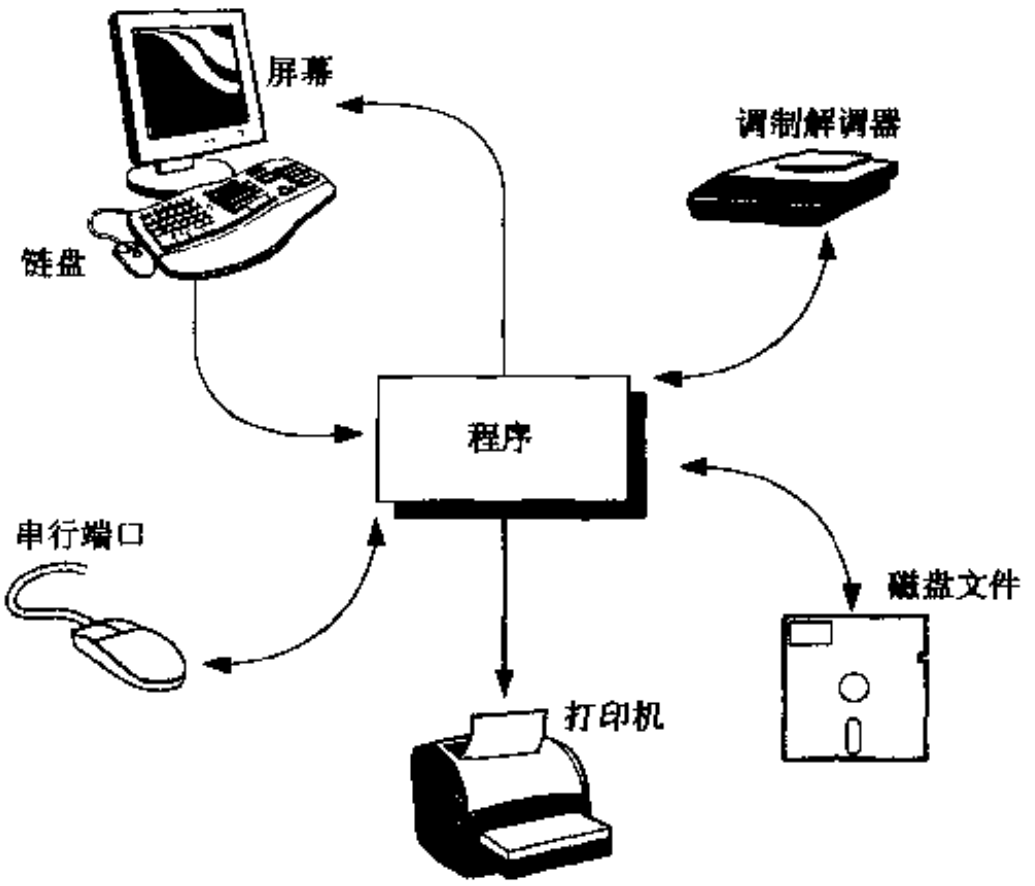


图 14.1　可在程序和各种外围设备之间进行输入和输出

每个流都与一个文件相连。这里的文件指的并不是磁盘文件，而是程序处理的流和用于输入/输出的设备之间的桥梁。大多数情况下，C 语言初学者无需关心这些文件，因为流、文件和设备之间的交互是由库函数和操作系统自动处理的。

14.1.3　文本流和二进制流

流的模式有两种：文本模式和二进制模式。文本流只包含字符，如发送给屏幕的文本数据。文本流被组织成行，其中每行最多可包含 255 个字符，并以换行符结尾。文本流中的有些字符（如换行符）有特殊含义。今天的课程只处理文本流。

二进制流能够处理各种类型的数据，包括（但不限于）文本数据。对于二进制流中的数据字节，不会以任何特殊的方式进行转换或解释，而只是按原来的样子被读写。二进制流主要用于读写磁盘文件。有关磁盘文件的内容将在第 16 天的课程中介绍。

14.1.4　预定义的流

ANSI 标准有三个预定义的流——也叫标准输入/输出文件。如果编写的程序将在运行 Windows 或 Dos 的 IBM 兼容 PC 上运行，则可能还有另外两个标准流可用。C 程序开始执行时，这些流将自动被打开；程序结束后，则自动被关闭。程序员无需采取任何措施来使这些流可用。表 14.1 列出了这些标准流以及它们连接的设备。这 5 个标准流都是文本模式的。

表 14.1　　5 个标准流

名　称	流	连接的设备
stdin	标准输入	键盘
stdout	标准输出	屏幕
stderr	标准错误	屏幕
stdprn*	标准打印机	打印机（LPT1:）
stdaux*	标准辅助	串行端口（COM1:）

*只在 DOS 和 Windows 下支持，而不是 ASNI 标准的一部分

实际上，您已经使用过其中的两个标准流。每当您使用函数 printf()或 puts()来将文本显示到屏幕上时，您使用了标准输出流；同样，当您使用 gets()或 scanf()来从键盘读取输入时，您使用了标准输入流。标准流

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

会被自动打开，而其他流（如用于操纵磁盘中的信息的流）必须显式地打开（第 16 天的课程将介绍如何打开）。本章后面的内容将介绍标准流。

14.2　使用 C 语言的流函数

C 语言的标准库包含很多处理流输入和输出的函数，其中大多数有两种版本：一种使用一个标准流；而另一种要求程序员指定一个流。表 14.2 列出了这些函数。该表并没有列出所有的输入/输出函数，另外，今天的课程也不会介绍表中列出的所有函数。

表 14.2　　标准库中的流输入/输出函数

使用一个标准流	需要指定一个流名	描　　述
printf()	fprintf()	格式化输出
vprintf()	vfprintf()	使用变量参数列表格式化输出
puts()	fputs()	输出字符串
putchar()	putc()、fputc()	输出字符
scanf()	fscanf()	格式化输入
vscanf()	vfscanf()	使用变量参数列表格式化输入
gets()	fgets()	输入字符串
getchar()	getc(), fgetc()	输入字符
perror()		将字符串输出到 stderr

要使用上述任何一个函数，都必须包含头文件 stdio.h；另外，要使用 perror()，还必须包含头文件 stdlib.h，而使用函数 vprintf()和 vfprintf()，还必须包含头文件 stdargs.h。在 UNIX 系统中，要使用 vprintf()和 vfprintf()，可能还需要包含头文件 varargs.h。编译器中的 Library Reference 将指出需要包含哪些头文件。

14.2.1　例子

程序清单 14.1 中的小程序演示了流的等效性。

程序清单 14.1　　stream.c：流的等效性

```
1:  /* Demonstrates the equivalence of stream input and output. */
2:  #include <stdio.h>
3:
4:  int main( void )
5:  {
6:     char buffer[256];
7:
8:     /* Input a line, then immediately output it. */
9:
10:    puts(gets(buffer));
11:
12:    return 0;
13: }
```

第 10 行使用 gets()函数从键盘（stdin）读取一行文本。由于 gets()返回一个指向字符串的指针，因此可将其用作 puts()的参数，而 puts()将该字符串显示到屏幕（stdout）上。该程序运行时，读取用户输入的一行文本，然后立刻将它显示到屏幕上。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

应　该	不应该
一定要使用 C 语言提供的标准输入/输出流。	不要修改或重命名标准流。 不要将标准输入流（如 stdin）用于输出函数（如 fprintf()）。

就 C 语言初学者而言，使用 gets()很不错。然而，在实际的程序中，应使用 fgets()（将在今天课程的后面介绍），因为 gets()会给程序带来安全隐患。

14.3　读取键盘输入

大多数 C 程序都需要从键盘（即 stdin 流）读取输入。输入函数分为三类：字符输入、行输入和格式化输入。

14.3.1　字符输入

字符输入函数每次从流中读取一个字符。当这些函数被调用时，将返回下一个字符或 EOF（如果到达文件末尾或发生错误）。EOF 是头文件 stdio.h 中定义的一个符号常量，值为-1。可以根据是否缓冲和回显来将字符输入函数分类：

- 有些字符输入函数进行缓冲。这意味着操作系统把所有的字符保存在一个临时存储空间内，直到用户按下 Enter 键后，再将它们发送到 stdin 流。其他一些不进行缓冲，这意味着每当用户按下一个键后，相应的字符将被立刻发送到 stdin 流。
- 有些输入函数将每个字符回显到 stdout。有些不回显，而只是将字符发送到 stdin，而不是 stdout。由于 stdout 对应的是屏幕，因此输入将被回显到屏幕上。

接下来的各节分别介绍缓冲、不缓冲、回显和不回显字符输入的用途。

1. getchar()函数

函数 getchar()读取 stdin 流中的下一个字符，它具备缓冲和回显功能。该函数的原型如下：

```
int getchar(void);
```

程序清单 14.2 演示了 getchar()的用法。putchar()函数（将在本章后面详细介绍）将一个字符显示到屏幕上。

程序清单 14.2　　getchar.c：getchar()函数

```
1: /* Demonstrates the getchar() function. */
2:
3: #include <stdio.h>
4:
5: int main( void )
6: {
7:     int ch;
8:
9:     while ((ch = getchar()) != '\n' )
10:         putchar(ch);
11:
12:    return 0;
13: }
```

该程序的运行情况如下：

```
This is what's typed in.
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
This is what's typed in.
```

第 9 行调用 getchar()函数从 stdin 读取一个字符。由于 getchar()是一个缓冲输入函数，因此在用户按下 Enter 键之前，该函数不会收到任何字符。但是，用户输入的每个字符都会被回显到屏幕上。

用户按下 Enter 键后，之前输入的所有字符（包括换行符）都将被操作系统发送给 stdin。getchar()函数每次返回一个字符，而该字符又被赋给 ch。

程序对每个字符进行判断，看它是否为换行符。如果不是，则使用 putchar()将其显示到屏幕上。当 getchar()返回换行符时，while 循环将终止。

getchar()函数可用来输入整行文本，如程序清单 14.3 所示；但对于这样的任务，其他函数更合适。

程序清单 14.3　　getchar2.c：使用 getchar()函数输入一整行文本

```
1:  /* Using getchar() to input strings. */
2:
3:  #include <stdio.h>
4:
5:  #define MAX 80
6:
7:  int main( void )
8:  {
9:      char ch, buffer[MAX+1];
10:     int x = 0;
11:
12:     while ((ch = getchar()) != '\n' && x < MAX)
13:         buffer[x++] = ch;
14:
15:     buffer[x] = '\0';
16:
17:     printf("%s\n", buffer);
18:
19:     return 0;
20: }
```

该程序的运行情况如下：

```
This is a string
This is a string
```

分析：该程序使用 getchar()的方式与程序清单 14.2 类似，但在循环中另外添加了一个条件。while 循环使用 getchar()读取字符，直到遇到换行符或读取的字符总数达到 80 个。读取的每个字符都被赋给数组 buffer。读取完字符后，第 15 行在数组的末尾加上一个空字符，以便第 17 行的 printf()函数能够打印输入的字符串。

第 9 行为何将 buffer 数组的长度声明为 MAX+1，而不是 MAX 呢？将 buffer 的长度声明为 MAX+1 时，该数组最多可包含 80 个字符和一个结尾的空字符。别忘了在字符串的末尾加上空字符。

2. getch()函数

getch()函数读取 stdin 流中的下一个字符，它不对输入字符进行缓冲，也不回显。getch()函数并非 ANSI 标准的一部分，这意味着并非在所有系统中都可以使用它。另外，为使用该函数，还可能需要包含其他的头文件。通常，getch()的原型位于头文件 conio.h 中，该原型如下：

```
int getch(void);
```

由于不对输入进行缓冲，因此用户按键后，getch()便立刻返回相应的字符，而无需等到用户按下 Enter 键。由于 getch()不回显输入，因此输入的字符不会显示到屏幕上。程序清单 14.4 演示了 getch()的用法。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com



警告：下面的程序清单使用了 getch()，而该函数不是 ANSI 标准的一部分。使用非 ANSI 函数时，一定要小心，因为并非所有的编译器都支持该函数。如果编译下述程序清单时发生错误，则很可能是因为您的编译器不支持 getch()。

程序清单 14.4　　getch.c：使用 getch()函数

```
1:  /* Demonstrates the getch() function. */
2:  /* Non-ANSI code */
3:  #include <stdio.h>
4:  #include <conio.h>
5:
6:  int main( void )
7:  {
8:      int ch;
9:
10:     while ((ch = getch()) != '\r')
11:         putchar(ch);
12:
13:     return 0;
14: }
```

该程序的运行情况如下：

```
Testing the getch() function
```

分析：运行该程序时，用户按键后，getch()便立刻返回相应的字符，而不会等到用户按下 Enter 键。该函数不回显，因此字符被显示到屏幕上是由于调用了 putchar()。为更深入地理解 getch()的工作原理，可在第 10 行的末尾加上一个分号，并删除第 11 行（putchar(ch)）。这样，当再次运行该程序时，您输入的内容不会被显示到屏幕上。getch()函数读取字符，但不把它们显示到屏幕上。您之所以知道字符被读取，是由于原来的程序清单使用了 putchar()来显示它们。

该程序为何将每个字符同\r（而不是\n）进行比较呢？\r 是一个转义序列，它表示回车。当用户按下 Enter 键后，键盘设备将一个回车发送给 stdin。对输入进行缓冲的字符输入函数会自动把回车转换为换行符，因此程序必须检测字符是否为\n，来确定用户是否按下了 Enter 键。不对输入进行缓冲的字符输入函数不进行这种转换，因此回车仍然为\r，因此程序必须检测\r。

程序清单 14.5 使用 getch()输入一整行文本。该程序的运行情况表明，getch()不回显输入。除了将 getchar()替换为 getch()外，该程序与程序清单 14.3 几乎完全相同。

程序清单 14.5　　getch2.c：使用 getch()输入一整行文本

```
1:  /* Using getch() to input strings. */
2:  /* Non-ANSI code */
3:  #include <stdio.h>
4:  #include <conio.h>
5:
6:  #define MAX 80
7:
8:  int main( void )
9:  {
10:     char ch, buffer[MAX+1];
11:     int x = 0;
12:
13:     while ((ch = getch()) != '\r' && x < MAX)
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
14:         buffer[x++] = ch;
15:
16:     buffer[x] = '\0';
17:
18:     printf("%s", buffer);
19:
20:     return0;
21: }
```

该程序的运行情况如下：

```
Here's a string
Here's a string
```



警告：前面说过，getch()不是 ANSI 标准命令。这意味着您的编译器（和其他编译器）可能支持，也可能不支持它。Symantec 和 Borland 的编译器支持 getch()，而 Microsoft 的编译器支持的是_getch()。如果您使用该函数时出现了问题，则应检查您的编译器是否支持 getch()。如果程序的可移植性很重要，则不应使用非 ANSI 函数。本书提供的 Bloodshed Dev-C++编译器支持 getch()，但是它的支持方式与上述编译器的不同。

3. getche()函数

本节的篇幅很短，因为 getche()类似于 getch()，只是它会将字符回显到 stdout。请修改程序清单 14.4，用 getche()代替 getch()。这样，当运行程序时，您输入的每个字符都会被显示到屏幕上两次：一次由 getche()回显，一次由 putchar()显示。



警告：getche()不是 ANSI 标准命令，但很多编译器支持它。

4. getc()和 fgetc()函数

字符输入函数 getc()和 fgetc()不会自动使用 stdin，而是让程序指定输入流。它们主要用于从磁盘文件中读取字符，更详细的情况请参阅第 16 天的课程。

应　该	不应该
一定要理解回显输入和非回显输入之间的差别；	如果可移植性很重要，则不要使用非 ANSI 标准函数；
一定要理解缓冲输入和非缓冲输入之间的差别。	在非-ANSI 标准函数的运行情况可能随使用的编译器而异。

5. 使用 ungetc()恢复一个字符

“恢复（ungetting）”一个字符是什么意思呢？下面通过一个例子来说明。假设程序从输入流中读取字符，且只能在读取输入末尾后的一个字符后，才能知道到达了输入末尾。例如，您只输入数字，因此遇到第一个非数字字符后，您便知道已到达输入的末尾。然而，第一个非数字字符可能是后续数据的重要组成部分，但已经将它从输入流中删除了。该字符丢失了吗？没有，可以将它“恢复”（返回到输入流中）。这样下次对该流执行输入操作时，首先读取的是这个字符。

要“恢复”一个字符，可以使用库函数 ungetc()。该函数的原型如下：

```
int ungetc(int ch, FILE *fp);
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

其中，参数 ch 是要恢复的字符，*fp 指定将字符归还到哪个流（可以是任何输入流）中。就现在而言，您只需将第二个参数指定为 stdin 即可：ungetc(ch, stdin);。FILE *fp 用于与磁盘文件相关联的流，这将在第 16 天的课程中介绍。

在两次读取操作之间，您只能恢复一个字符，但任何时候都不能恢复 EOF。如果恢复成功，该函数返回 ch；否则返回 EOF。

6. 行输入

行输入函数从输入流中读取一行——读取换行符之前的所有字符。标准库中包含两个行输入函数：gets() 和 fgets()。

7. gets()函数

第 10 天的课程已经介绍过 gets()函数。该函数很简单，它从 stdin 中读取一行，并将它存储在一个字符串中。该函数的原型如下：

```
char *gets(char *str);
```

您可能已经理解了该原型的含义。gets()将一个 char 指针作为参数，并返回一个 char 指针。gets()函数读取 stdin 中的字符，直到到达换行符（\n）或文件末尾，然后将换行符替换为空字符，并将字符串存储到 str 指定的位置。

返回值是一个指向字符串的指针（str）。如果 gets()发生错误或读取任何字符之前就到达了文件末尾，则返回一个空指针。

调用 gets()之前，必须分配足够的内存空间来存储字符串，分配的方法已经在第 10 天的课程中介绍过了。该函数无法知道 ptr 指向的空间是否是分配了的，而是直接将字符串存储到 ptr 指向的位置。如果该空间没有分配，则字符串可能覆盖其他数据，从而导致程序错误。

程序清单 10.5 和 10.6 使用了 gets()。

8. fgets()函数

库函数 fgets()与 gets()类似，也从输入流中读取一行文本；但它更灵活，因为它允许程序员指定输入流以及最多读取的字符数。fgets()常用于输入磁盘文件中的文本，这将在第 16 天的课程中介绍。要使用它来读取 stdin 中的输入，可以将输入流指定为 stdin。fgets()的原型如下：

```
char *fgets(char *str, int n, FILE *fp);
```

最后一个参数（FILE *fp）用于指定输入流。就现在而言，只需将其指定为标准输入流（stdin）即可。

指针 str 用于指定输入字符串的存储位置。参数 n 指定最多输入的字符数。fgets()函数从输入流中读取字符，直到遇到换行符、文件末尾或者已经读取了 n-1 个字符。被存储的字符串中包含换行符，并以空字符结尾。fgets()的返回值与 gets()相同。

严格地说，fgets()并不一定读取一行文本（如果一行指的是一个以换行符结尾的字符序列）。如果一行中包含的字符超过了 n-1 个，则它不会读取整行。用于读取 stdin 中的输入时，仅当用户按下 Enter 键后，fgets()函数才结束运行，但它只将前 n-1 个字符存储到字符串中。仅当一行中包含的字符少于 n-1 个时，换行符才被存储到字符串中。程序清单 14.6 演示了 fgets()函数的用法。

程序清单 14.6　　fgets.c：使用 fgets()读取键盘输入

```
1:  /* Demonstrates the fgets() function. */
2:
3:  #include <stdio.h>
4:
5:  #define MAXLEN 10
6:
7:  int main( void )
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
8:  {
9:     char buffer[MAXLEN];
10:
11:    puts("Enter text a line at a time; enter a blank to exit.");
12:
13:    while (1)
14:    {
15:        fgets(buffer, MAXLEN, stdin);
16:
17:        if (buffer[0] == '\n')
18:           break;
19:
20:        puts(buffer);
21:    }
22:    return 0;
23: }
```

该程序的运行情况如下：

```
Enter text a line at a time; enter a blank to exit.
Roses are red
Roses are
 red
Violets are blue
Violets a
re blue

Programming in C
Programmi
ng in C

Is for people like you!
Is for pe
ople like
 you!
```

第 15 行使用了 fgets()函数。运行该程序时，分别输入多于和少于 MAXLEN 个字符，看看发生的情况。如果输入的字符数多于 MAXLEN，则第一次调用 fgets()时，只读取前 MAXLEN-1 个字符，其余的字符将留在键盘缓冲区中，下次调用 fgets()或其他读取 stdin 的函数时，将读取这些字符。如果用户输入一个空行，则程序将终止（第 17 和 18 行）。

14.3.2　格式化输入

至此介绍的所有输入函数都只是从输入流中读取一个或多个字符，然后将它们存储到内存中，而没有对输入进行解释或格式化，因此无法将输入存储到数值变量中。例如，如何从键盘读取 12.86，并将其赋给一个 float 变量呢？可以使用 scanf()和 fscanf()函数。第 7 天的课程介绍了 scanf()函数，本节将更详细地介绍其用法。

这两个函数基本相同，只是 scanf()总是使用 stdin，而在 fscanf()函数中，程序员可以指定输入流。本节只介绍 scanf()函数，而 fscanf()通常用于读取磁盘文件，将在第 16 天的课程中介绍。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

1. scanf()函数的参数

scanf()函数接受多种参数，但有两个参数是必不可少的。第一个参数是格式化字符串，它使用特殊字符告诉 scanf()函数如何解释输入；第二个及其他的参数是变量的地址，输入的数据将被存储在这些地址中。下面是一个例子：

```
scanf("%d", &x);
```

其中第一个参数"%d"是格式化字符串，命令 scanf()查找一个有符号整型值。第二个参数使用地址运算符（&）来命令 scanf()将输入值赋给变量 x。下面详细介绍格式化字符串。

scanf()中的格式化字符串可以包含以下内容：

- 空格和制表符（用来提高格式化字符串的可读性），它们将被忽略；
- 字符（除了%），用于同输入中的非空白字符进行匹配；
- 一个或多个转换说明，由字符%和特殊字符组成。通常，对于每个变量，格式化字符串中都包含一个相应的转换说明。

格式化字符串中，唯一必不可少的是转换说明。每个转换说明都以%打头，并包含可选的和必选的内容。scanf()依次将格式化字符串中的转换说明用于输入字段。输入字段是一个由非空白字符组成的序列，到达指定的宽度或空白后，输入字段便结束。转换说明包含以下内容：

- 位于%后面的、可选的赋值抑制标记（*）。该字符命令 scanf()按当前的转换说明符进行转换，但忽略转换后的结果（即不将它赋给任何变量）；
- 字段宽度（可选）。字段宽度是一个十进制数，用于指定输入字段的宽度（单位为字符）。换句话说，字段宽度指定 scanf()在执行当前的转换时，应考虑 stdin 中的多少个字符。如果没有指定字段宽度，则以下一个空白作为字段结束标记。
- 精度限定符（可选）。是一个字符，可以是 h、l 或 L。精度限定符改变它后面的类型说明符的含义，有关细节将在本章后面介绍。
- 类型说明符（必不可少）。类型说明符是一个或多个字符，它告诉 scanf()如何对输入进行解释。表 14.3 列出这些字符，并对它们做了描述。其中，“参数”列出了相应的变量类型。例如，类型说明符 d 对应于 int * 变量（int 指针）。

表 14.3　　scanf()的转换说明符中使用的类型说明符

类　型	参　数	含　义
d	int *	十进制整数
i	int *	十进制、八进制（以 0 开头）或十六进制（以 0x 或 0X 开头）整数
o	int *	八进制整数（以 0 打头，也可以不以 0 打头）
u	unsigned int *	无符号十进制整数
x	int *	十六进制整数（以 0x 或 0X 打头，也可以不以它们打头）
c	char *	读取一个或多个字符，并将它们依次存储到参数指定的内存单元中。不在末尾加上空字符。如果没有指定字段宽度，则只读取一个字符；否则读取指定数目的字符（包括空白）。
s	char *	将一个由非空白字符组成的字符串存储到指定的内存单元中，并在末尾加上空字符。
a,e,f,g	float *	浮点数。输入该数字时，可以使用科学计数法，也可以使用小数点表示法。
[...]	char *	一个字符串。只读取方括号中列出的字符。遇到不匹配的字符、达到指定的字段宽度或用户按 Enter 键后，立刻停止读取。为读取字符]，应首先将它列出：[]...]。将在字符串末尾加上空字符。
[^...]	char *	与[...]相同，不过只读取方括号中没有列出的字符。
%	None	读取%字符，且不执行任何赋值操作。

介绍使用 scanf()的范例前，先介绍一下精度限定符，如表 14.4 所示。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

表 14.4　精度限定符

精度限定符	含　义
hh	用于类型说明符 d、i、o、u、x、X 或 n 前面时，指定参数是指向 signed char 或 unsigned char 的指针。
h	用于类型说明符 d、i、o、u、x、X 或 n 前面时，指定参数是指向 shor int 或 unsigned short int 的指针。
l	用于类型说明符 d、i、o、u、x、X 或 n 前面时，指定参数是指向 long 或 unsigned long 的指针；用于类型说明符 a、A、e、E、f、F、g 或 G 前面时，指定参数是指向 double 的指针。
ll	用于类型说明符 d、i、o、u、x、X 或 n 前面时，指定参数是指向 long long 或 unsigned long long 的指针；
L	用于类型说明符 a、A、e、E、f、F、g 或 G 前面时，指定参数是指向 long double 的指针。

2. 处理多余的字符

使用 scanf()读取输入时，输入被缓冲——用户按下 Enter 键之前，不会从 stdin 那里收到任何字符。用户按下 Enter 键后，scanf()将从 stdin 那里收到整行字符，并依次对它们进行处理。仅当收到的字符足够与格式化字符串中的转换说明匹配后，scanf()才结束运行。另外，scanf()也只处理格式化字符串指定的字符数目。多余的字符仍保留在 stdin 中。这些字符可能导致问题。下面详细介绍 scanf()的工作原理，解释为何会导致问题。

调用 scanf()后，用户输入了一行数据，结果有三种。假设执行的是 scanf("%d %d", &x, &y);，换句话说，scanf()期望用户输入两个十进制整数。可能的情况有三种：

- 用户输入的内容与格式化字符串匹配。例如，假设用户输入 12 14，然后按 Enter 键。在这种情况下，输入的数据能够满足 scanf()的要求，也没有在 stdin 中留下任何字符。
- 用户输入的数据太少，与格式化字符串不匹配。例如，假设用户输入 12，然后按 Enter 键。在这种情况下，scanf()继续等待缺少的输入。收到所需的输入后，程序将继续运行，同时没有在 stdin 中留下多余的字符。
- 用户输入的数据多于格式化字符串要求的。例如，假设用户输入 12 14 16，然后按 Enter 键。在这种情况下，scanf()读取 12 和 14，然后返回。多余的字符 1 和 6 将留在 stdin 中。

第三种情况（即余下的字符）可能导致问题。在程序运行期间，它们一直留在 stdin 中，直到程序下一次从 stdin 中读取输入。此时，首先读取这些余下的字符，然后再读取当前用户输入的数据。导致问题的原因显而易见。例如，下面的代码要求用户首先输入一个整数，然后输入一个字符串：

```
puts("Enter your age.");
scanf("%d", &age);
puts("Enter your first name.");
scanf("%s", name);
```

假设第一次输入时，用户为提高精度输入了 29.00，然后按 Enter 键。第一个 scanf()调用期望一个整数，因此它读取 stdin 中的 29，并将它赋给变量 age。这样，字符.00 将保留在 stdin 中。第二个 scanf()调用期望一个字符串，它在 stdin 中查找，并找到.00。结果是，.00 被赋给了 name。

如何避免这种问题呢？一种解决方案是希望用户输入信息时不犯错误，但这是不现实的。

一种更好的解决方案是，提示用户输入之前，确保 stdin 中没有多余的字符。为此，可以调用 gets()，该函数读取换行符之前的所有字符（包括换行符）。您可以在一个单独的函数中调用 gets()，并给它取一个描述性名称 clear_kb()，而不是直接在程序中调用 gets()，如程序清单 14.7 所示。

程序清单 14.7　clearing.c：清除 stdin 中多余的字符，以免发生错误

```
1:  /* Clearing stdin of extra characters. */
2:
3:  #include <stdio.h>
4:
5:  void clear_kb(void);
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
6:
7:  int main( void )
8:  {
9:     int age;
10:     char name[20];
11:
12:     /* Prompt for user's age.*/
13:
14:     puts("Enter your age.");
15:     scanf("%d", &age);
16:
17:     /* Clear stdin of any extra characters. */
18:
19:     clear_kb();
20:
21:     /* Now prompt for user's name:*/
22:
23:     puts("Enter your first name:");
24:     scanf("%s", name);
25:     /* Display the data. */
26:
27:     printf("Your age is %d.\n", age);
28:     printf("Your name is %s.\n", name);
29:
30:     return 0;
31: }
32:
33: void clear_kb(void)
34:
35: /* Clears stdin of any waiting characters. */
36: {
37:     char junk[80];
38:     gets(junk);
39: }
```

该程序的运行情况如下：

```
Enter your age:
29 and never older!
Enter your first name:
Bradley
Your age is 29.
Your name is Bradley.
```

分析：运行该程序时，请在输入年龄之后输入其他一些字符，再按 Enter 键。确保程序将忽略它们，并能正确地显示您的姓名。然后，对程序进行修改，删除其中对 clear_kb ()的调用，并再次运行程序。您将发现，随年龄一起输入的字符将被赋给 name。

3. 使用 fflush()处理多余的字符

还有另一种清除多余字符的方法。fflush()函数冲洗流（包括标准输入流）中的信息，该函数通常用于处理磁盘文件(将在第 16 天的课程中介绍)，但也可以使用它来简化程序清单 14.7。程序清单 14.8 使用了 fflush()函数，而不是程序清单 14.7 中创建的 clear_kb()函数。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 14.8　　clear.c: 使用 fflush()来清除 stdin 中多余的字符

```
1:  /* Clearing stdin of extra characters. */
2:  /* Using the fflush() function         */
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:      int age;
8:      char name[20];
9:
10:     /* Prompt for user's age.*/
11:     puts("Enter your age.");
12:     scanf("%d", &age);
13:
14:     /* Clear stdin of any extra characters. */
15:     fflush(stdin);
16:
17:     /* Now prompt for user's name.*/
18:     puts("Enter your first name.");
19:     scanf("%s", name);
20:
21:     /* Display the data. */
22:     printf("Your age is %d.\n", age);
23:     printf("Your name is %s.\n", name);
24:
25:     return 0;
26: }
```

该程序的运行情况如下：

```
Enter your age.
29 and never older!
Enter your first name.
Bradley
Your age is 29.
Your name is Bradley.
```

分析：第 15 行使用了 fflush()函数，该函数的原型如下：

```
int fflush(FILE *stream);
```

其中 stream 是要冲洗的流。在程序清单 14.8 中，传递给 fflush()的是标准输入流 stdin。

4. scanf()应用范例

要熟悉 scanf()函数的工作原理，最好的方式是使用它。这个函数功能强大，但常常令人迷惑。请尝试使用它，并看看结果。程序清单 14.9 演示了 scanf()的一些不太常见的用法。您应编译并运行该程序，然后尝试对 scanf()的格式化字符串进行修改，看看发生的情况。

程序清单 14.9　　scanf.c: 一些使用 scanf()读取键盘输入的方式

```
1:  /* Demonstrates some uses of scanf(). */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
6:  {
7:      int i1;
8:      int i2;
9:      long l1;
10:
11:     double d1;
12:     char buf1[80]
13:     char buf2[80];
14:
15:     /* Using the l modifier to enter long integers and doubles.*/
16:
17:     puts("Enter an integer and a floating point number.");
18:     scanf("%ld %lf", &l1, &d1);
19:     printf("\nYou entered %ld and %lf.\n",l1, d1);
20:     puts("The scanf() format string used the l modifier to store");
21:     puts("your input in a type long and a type double.\n");
22:
23:     fflush(stdin);
24:
25:     /* Use field width to split input. */
26:
27:     puts("Enter a 5 digit integer (for example, 54321).");
28:     scanf("%2d%3d", &i1, &i2);
29:
30:     printf("\nYou entered %d and %d.\n", i1, i2);
31:     puts("Note how the field width specifier in the scanf() format");
32:     puts("string split your input into two values.\n");
33:
34:     fflush(stdin);
35:
36:     /* Using an excluded space to split a line of input into */
37:     /* two strings at the space. */
38:
39:     puts("Enter your first and last names separated by a space.");
40:     scanf("%[^ ]%s", buf1, buf2);
41:     printf("\nYour first name is %s\n", buf1);
42:     printf("Your last name is %s\n", buf2);
43:     puts("Note how [^ ] in the scanf() format string, by excluding");
44:     puts("the space character, caused the input to be split.");
45:
46:     return 0;
47: }
```

该程序的运行情况如下：

```
Enter an integer and a floating point number.
123 45.6789

You entered 123 and 45.678900.
The scanf() format string used the l modifier to store
your input in a type long and a type double.

Enter a 5 digit integer (for example, 54321).
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
54321

You entered 54 and 321.
Note how the field width specifier in the scanf() format
string split your input into two values.

Enter your first and last names separated by a space.
Gayle Johnson

Your first name is Gayle
Your last name is Johnson
Note how [^ ] in the scanf() format string, by excluding
the space character, caused the input to be split.
```

分析：该程序首先定义了一些变量（第 7～13 行），用于存储输入的数据。然后，让用户输入各种类型的数据。第 17～21 行让用户输入打印一个长整型值和一个 double 值。第 23 行调用 fflush()函数来清除标准输入流中的字符。第 27 和 28 行读取下一个值：一个五位的整数。由于提供了宽度说明符，因此这个五位的整数被拆分成两个整数：一个两位，一个三位。第 34 行再次调用 fflush()来冲洗标准输入流。最后，第 36～44 行使用了排除字符的功能。第 40 行使用"%[^]"来命令 scanf()读取空白之前的字符；这相当于将输入拆分开来。

请花些时间修改该程序清单，并输入其他的值，看看结果如何。

scanf()可满足大部分输入需求，尤其是涉及到数字的输入（对于输入字符串，使用 gets()更容易）。不过，您有必要自己编写一些专用的输入函数。有关用户定义的函数范例，请参阅第 18 天的课程。

应　该	不应该
只使用标准输入流时，一定要使用 gets()和 scanf()，而不要使用 fgets()和 fscanf()函数。	别忘了清除输入流中多余的字符。

14.4　控制屏幕输出

与输入函数一样，屏幕输出函数也分为三大类：字符输出函数、行输出函数和格式化输出函数。前面的课程已经介绍过一些这样的函数，本节将对它们做更详细的讨论。

14.4.1　使用 putchar()、putc()和 fputc()输出字符

C 语言库中的字符输出函数将一个字符发送给流。函数 putchar()将输出发送给 stdout（通常是屏幕）；函数 fputs()和 putc()将输出发送到参数列表中指定的流。

1．使用 putchar()函数

putchar()函数的原型如下，它位于 stdio.h 中：

```
int putchar(int c);
```

该函数将存储在变量 c 中的字符写入到 stdout。虽然原型中指定的参数类型为 int，但可以将 char 变量传递给 putchar()，也可以将 int 变量传递给它（只要其值对应于一个字符，即取值范围为 0～255）。该函数返回写入到 stdout 中的字符；如果发生错误，则返回 EOF。

程序清单 14.2 已经演示过 putchar()。程序清单 14.10 显示 ASCII 值为 14～127 的字符。

程序清单 14.10　　putchar.c：使用 putchar()函数

```
1:  /* Demonstrates putchar(). */
2:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
3:  #include <stdio.h>
4:  int main( void )
5:  {
6:    int count;
7:
8:    for (count = 14; count < 128; )
9:         putchar(count++);
10:
11:   return 0;
12: }
```

该程序的输出如下：

```
?□????¶§???????????? !"#$%&'()*+,-./0123456789:;<=>?@ABCDEF
GHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

也可以使用 putchar()函数来显示字符串（如程序清单 14.11 所示），虽然其他函数更适合完成这种任务。

程序清单 14.11　　putchar2.c：使用 putchar()显示字符串

```
1:  /* Using putchar() to display strings. */
2:
3:  #include <stdio.h>
4:
5:  #define MAXSTRING 80
6:
7:  char message[] = "Displayed with putchar().";
8:  int main( void )
9:  {
10:    int count;
11:
12:    for (count = 0; count < MAXSTRING; count++)
13:    {
14:
15:       /* Look for the end of the string. When it's found, */
16:       /* write a newline character and exit the loop. */
17:
18:       if (message[count] == '\0')
19:       {
20:          putchar('\n');
21:          break;
22:       }
23:       else
24:
25:       /* If end of string not found, write the next character. */
26:
27:          putchar(message[count]);
28:    }
29:    return0;
30: }
```

该程序的输出如下：

```
Displayed with putchar().
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

2. putc()和 fputc()函数

putc()和 fputc()函数的功能相同：将一个字符发送给指定的流。putc()以宏的方式实现了 fputs()，有关宏将在第 21 天的课程中介绍。就现在而言，只需坚持使用 fputc()即可。该函数的原型如下：

```
int fputc(int c, FILE *fp);
```

您可能对其中的 FILE *fp 感到迷惑，该参数用于将输出流传递给 fputc()（更详细的信息，请参阅第 16 天的课程）。如果将输出流指定为 stdout，则 fputs()的行为与 putchar()完全相同。因此下面两条语句等价：

```
putchar('x');
fputc('x', stdout);
```

14.4.2 使用 puts()和 fputs()输出字符串

程序在屏幕上显示的更多的是字符串，而不是单个字符。库函数 puts()用于显示字符串。函数 fputs()将字符串发送给指定的流，除此之外，其他方面与 puts()完全相同。puts()函数的原型如下：

```
int puts(char *cp);
```

其中 cp 是一个指针，指向要显示的字符串的第一个字符。puts()函数显示结尾的空字符之前的所有字符（但不包括空字符），并自动换行。如果显示成功，则返回一个正值；否则返回 EOF（EOF 是 stdio.h 中定义的一个符号常量，值为-1）。

puts()函数可用于显示各种字符串，如程序清单 14.12 所示。

程序清单 14.12　　puts.c：使用 puts()函数显示字符串

```
1:  /* Demonstrates puts(). */
2:
3:  #include <stdio.h>
4:
5:  /* Declare and initialize an array of pointers. */
6:
7:  char *messages[5] = { "This", "is", "a", "short", "message." };
8:
9:  int main( void )
10: {
11:     int x;
12:
13:     for (x=0; x<5; x++)
14:         puts(messages[x]);
15:
16:     puts("And this is the end!");
17:
18:     return 0;
19: }
```

该程序的输出如下：

```
This
is
a
short
message.
And this is the end.
```

分析：该程序清单声明了一个指针数组（有关指针数组，将在明天的课程中介绍）。第 13 和 14 行打印存储在数组 message 中的每个字符串。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

14.4.3 使用 printf()和 fprintf()格式化输出

至此介绍的输出函数都只能显示字符和字符串。如何显示数字呢？要显示数字，必须使用 C 语言库中的格式化输出函数：printf()和 fprintf()。这些函数也能够显示字符和字符串。第 7 天的课程介绍过 printf()函数，从那以后，几乎每天的课程都使用过它。本节介绍有关该函数的其他细节。

函数 printf()和 fprintf()几乎相同，只是前者总是将输出发送给 stdout，而后者可以指定输出流。fprintf()通常用于将数据输出到磁盘文件中，这将在第 16 天的课程中介绍。

printf()接受可变数目的参数，但有一个参数是比不可少的。第一个参数（也是唯一一个必不可少的参数）是格式化字符串，它告诉 printf()如何格式化输出。可选的参数是您要显示其值的变量或表达式。介绍细节之前，先来看几个例子，以便您对 printf()有一定的感性认识。

- 语句 printf(“Hello, World.”);在屏幕上显示消息 “Hello, World”。这个例子只使用了一个参数：格式化字符串，而该字符串只包含一个要显示到屏幕上的字面字符串。
- 语句 prinf(“%d”, i);将整型变量 i 的值显示到屏幕上。其中，格式化字符串只包含格式说明符%d，它命令 printf()显示一个十进制整数。第二个参数 i 是要显示其值的变量的名称。
- 语句 printf(“%d plus %d equals %d.”, a, b, a+b);在屏幕上显示“2 plus 3 equals 5”（假设 a 和 b 都是整型变量，它们的值分别是 2 和 3）。其中有 4 个参数：格式化字符串（包含字面文本和格式说明符）以及两个变量和一个表达式（将显示它们的值）。

下面详细介绍 printf()的格式化字符串。它可以包含以下内容：

- 零个、一个或多个转换命令，告诉 printf()如何显示其参数列表中的值。转换命令由%和一个或多个字符组成。
- 除转换命令之外的字符，它们将按原样显示。

在前面的第三个例子中，格式化字符串为%d plus %d equals %d，其中的三个%d 为转换命令，其余的（包括空白）为字面字符，将按原样显示。

下面详细介绍转换命令。转换命令的组成如下，其中用方括号括起的是可选的：

```
%[flag][field_width][.[precision]][l]conversion_char
```

conversion_char 是转换命令中唯一必不可少的部分（%除外）。表 14.5 列出了转换字符，并对它们的含义做了说明。

表 14.5　printf()和 fprintf()使用的转换字符

转换字符	含　义
d, I	以十进制方式显示有符号整数
u	以十进制方式显示无符号整数
o	以八进制方式显示无符号整数
x, X	以十六进制方式显示无符号整数，x 表示输出小写，X 表示输出大写
c	显示一个字符（参数为该字符的 ASCII 码）
e, E	以科学计数法显示 float 或 double 值（例如 123.45 被显示为 1.234500e+002）。如果没有使用 f 说明符指定精度，则显示 6 位小数。e 和 E 用户控制输出的大小写。
f	以十进制计数法显示 float 或 double 值（例如，123.45 被显示为 123.450000）。如果没有指定精度，则显示 6 位小数。
g, G	使用 e、E 或 f 格式。如果指数小于-3 或大于精度（默认为 6），则使用 E 或 e 格式；否则使用 f 格式。末尾多余的 0 将被截除。
n	不显示任何东西。与转换命令 n 对应的参数为 int 指针。printf()将当前输出的字符数赋给该变量。
s	显示一个字符串。对应的参数为一个 char 指针。显示空字符之前的所有字符或指定数目的字符。不显示末尾的空字符。
%	显示字符%。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

可以在转换字符前加上限定符 l，该限定符只能用于转换字符 o、u、x、X、i、d 和 b，它指定参数类型为 long，而不是 int。将限定符 l 用于转换字符 e、E、f、g 或 G 时，它指定参数类型为 double。将 l 用于其他字符时，它会被忽略。

除了 l 外，还可以使用限定符 ll，后者的工作原理与前者相同，只是它将参数的类型指定为 long long 而不是 long。

精度限定符可以是小数点或小数点和数字，它只能用于转换字符 e、E、f、g、G 或 s，用于指定小数位数或输出的字符数（用于 s 时）。如果精度限定符只包含小数点，则精度为 0。

字段宽度限定符指定了输出的最小字符数，其内容如下：

- 不以 0 打头的整数。在输出左边添加空格，以满足指定的字段宽度。
- 以 0 打头的整数。在输出左边添加 0，以满足指定的字段宽度。
- *字符。下一个参数（其类型必须为 int）用于指定字段宽度。例如，如果 w 的类型为 int，值为 10，则语句 printf("%*d", w, a);打印 a 的值，并使其字段宽度为 10。

如果没有指定字段宽度或指定的宽度比输出窄，则字段宽度取决于需求。

printf()的格式化字符串的最后一个可选项是标记（flag），它位于字符%的后面，可能的 4 种值如下：

- -：输出是左对齐，而不是默认的右对齐；
- +：对于有符号数，在其左边加上+或-；
- ' '：在正数前面加上空格；
- #：只能用于转换字符 x、X 或 o。表示对于非 0 数字，在前面加上 0X、0x（对于 x 和 X）或 0（对于 o）。

使用 printf()时，格式化字符串可以是用双引号括起的字面字符串，也可以是存储在内存中的、以空字符结尾的字符串。对于后一种情况，您把一个指向字符串的指针传递给 printf()。例如，下述语句：

```
char *fmt = "The answer is %f.";
printf(fmt, x);
```

与下面的语句等价：

```
printf("The answer is %f.", x);
```

正如第 7 天指出的，printf()的格式化字符串可以包含转义序列，以对输出进行特殊控制。表 14.6 列出了最常用的转义序列。例如，换行符序列（\n）导致后续的输出出现在下一行。

表 14.6　　最常用的转义序列

序　列	含　义
\a	振铃
\b	回退
\n	换行
\t	水平制表符
\\	垂直制表符
\?	问号
\'	单引号
\"	双引号

printf()函数比较复杂。学习其用法的最佳方式是先看一些范例，然后进行实践。程序清单 14.13 演示了 printf()的很多用法。

程序清单 14.13　　printf.c：printf()函数的一些用法

```
1:  /* Demonstration of printf(). */
2:
3:  #include <stdio.h>
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
4:
5:  char *m1 = "Binary";
6:  char *m2 = "Decimal";
7:  char *m3 = "Octal";
8:  char *m4 = "Hexadecimal";
9:
10: int main( void )
11: {
12:     float d1 = 10000.123;
13:     int n, f;
14:
15:
16:     puts("Outputting a number with different field widths.\n");
17:
18:     printf("%5f\n", d1);
19:     printf("%10f\n", d1);
20:     printf("%15f\n", d1);
21:     printf("%20f\n", d1);
22:     printf("%25f\n", d1);
23:
24:     puts("\n Press Enter to continue...");
25:     fflush(stdin);
26:     getchar();
27:
28:     puts("\nUse the * field width specifier to obtain field width");
29:     puts("from a variable in the argument list.\n");
30:
31:     for (n=5;n<=25; n+=5)
32:        printf("%*f\n", n, d1);
33:
34:     puts("\n Press Enter to continue...");
35:     fflush(stdin);
36:     getchar();
37:
38:     puts("\nInclude leading zeros.\n");
39:
40:     printf("%05f\n", d1);
41:     printf("%010f\n", d1);
42:     printf("%015f\n", d1);
43:     printf("%020f\n", d1);
44:     printf("%025f\n", d1);
45:
46:     puts("\n Press Enter to continue...");
47:     fflush(stdin);
48:     getchar();
49:
50:     puts("\nDisplay in octal, decimal, and hexadecimal.");
51:     puts("Use # to precede octal and hex output with 0 and 0X.");
52:     puts("Use - to left-justify each value in its field.");
53:     puts("First display column labels.\n");
54:
55:     printf("%-15s%-15s%-15s", m2, m3, m4);
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
56:
57:     for (n = 1;n< 20; n++)
58:        printf("\n%-15d%-#15o%-#15X", n, n, n);
59:
60:     puts("\n Press Enter to continue...");
61:     fflush(stdin);
62:     getchar();
63:
64:     puts("\n\nUse the %n conversion command to count characters.\n");
65:
66:     printf("%s%s%s%s%n", m1, m2, m3, m4, &n);
67:
68:     printf("\n\nThe last printf() output %d characters.\n", n);
69:
70:     return 0;
71: }
```

该程序的输出如下：

```
Outputting a number with different field widths.

10000.123047
10000.123047
      10000.123047
            10000.123047
                  10000.123047

Press Enter to continue...

Use the * field width specifier to obtain field width
from a variable in the argument list.

10000.123047
10000.123047
      10000.123047
            10000.123047
                  10000.123047

  Press Enter to continue...

Include leading zeros.

10000.123047
10000.123047
00010000.123047
0000000010000.123047
000000000000010000.123047

Press Enter to continue...

Display in octal, decimal, and hexadecimal.
Use # to precede octal and hex output with 0 and 0X.
Use - to left-justify each value in its field.
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
First display column labels.

Decimal       Octal         Hexadecimal
1             01            0X1
2             02            0X2
3             03            0X3
4             04            0X4
5             05            0X5
6             06            0X6
7             07            0X7
8             010           0X8
9             011           0X9
10            012           0XA
11            013           0XB
12            014           0XC
13            015           0XD
14            016           0XE
15            017           0XF
16            020           0X10
17            021           0X11
18            022           0X12
19            023           0X13

Press Enter to continue...

Use the %n conversion command to count characters.

BinaryDecimalOctalHexadecimal

The last printf() output 29 characters.
```

14.5 重定向输入/输出

使用 stdin 和 stdout 的程序可以利用操作系统的特性——重定向。重定向让您能够：

- 将发送到 stdout 的输出重定向到磁盘文件或打印机。
- 让 stdin 的程序输入来自磁盘文件而不是键盘。

您无需在程序中编写重定向的代码，而是在运行程序时，在命令行中指定这一点。在 DOS 和 Microsoft Windows 的提示符下，用于重定向的符号与 UNIX 相同，都是<and>。下面首先讨论重定向输出。

还记得您编写的第一个 C 程序 hello.c 吗？它使用库函数 printf()在屏幕上显示消息 Hello, world。您知道，printf()将输出发送到 stdout，因此可对输出进行重定向。方法是，在命令行提示符下输入程序名的同时，加上符号>和新的目标名：

```
hello > destination
```

因此，如果您输入 hello > prn，则程序的输出将被发送到打印机，而不是屏幕（在 DOS 中，prn 指的是与端口 LPT1:相连的打印机）；如果您输入 hello > hello.txt，则输出将被存储到磁盘文件 hello.txt 中。

将输出重定向到磁盘文件时，一定要小心。如果该文件已经存在，则文件原来的内容会被新内容所覆盖；如果不存在，则将被创建。将输出重定向到文件时，也可以使用符号>>。在这种情况下，如果指定的文件已经存在，则输出将被附加到文件末尾。程序清单 14.14 演示了如何进行重定向。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

程序清单 14.14　　redirect.c：重定向输入/输出

```
1:  /* Can be used to demonstrate redirection of stdin and stdout. */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:    char buf[80];
8:
9:    gets(buf);
10:   printf("The input was: %s\n", buf);
11:   return 0;
12: }
```

分析：该程序从 stdin 读取一行输入，然后在前面加上 The input was:，并将其发送给 stdout。编译并链接该程序后，在命令行提示符下输入 redirect，以运行该程序，且不进行重定向。如果输入 I am teaching myself C，则程序将在屏幕上显示下述内容：

```
The input was: I am teaching myself C
```

如果通过执行命令 redirect >test.txt 来运行该程序，并输入同样的内容，则程序不会在屏幕上显示任何东西，而是在磁盘上创建一个名为 test.txt 的文件。如果使用 DOS 命令 TYPE 来显示该文件的内容：

```
type test.txt
```

您将发现其内容为：The input was: I am teaching myself C。同样，如果您执行命令 redirect >prn 来运行该程序，则将在打印机上打印输出行（prn 是 Dos 命令中打印机的名字）。

请再次运行该命令。这次使用>>符号将输出重定向到文件 TEST.TXT。这样，输出将附加到该文件的后面，而不会覆盖原有的内容。

14.5.1　重定向输入

现在介绍重定向输入。首先，需要一个提供输入的文件。请使用编辑器创建一个名为 INPUT.TXT 的文件，该文件包含一行内容：William Shakespeare。现在，在 DOS 提示符下执行下面的命令，运行程序清单 14.14：

```
redirect < INPUT.TXT
```

程序不会等待您从键盘输入，而是立刻在屏幕上显示如下消息：

```
The input was: William Shakespeare
```

stdin 流被重定向到 INPUT.TXT，因此程序调用 gets()时，将从该文件（而不是键盘）读取一行内容。

可以同时对输入和输出进行重定向。请执行下面的命令运行该程序，从而将 stdin 重定向到 INPUT.TXT，将 stdout 重定向到 JUNK.TXT：

```
redirect < INPUT.TXT >junk.txt
```

在有些情况下，重定向 stdin 和 stdout 很有用。例如，排序程序可以对键盘输入或磁盘文件的内容进行排序。同样，邮寄名单程序可以将地址显示在屏幕上，将它们发送给打印机打印邮寄地址签，或将其放在一个文件中以做他用。



注意：重定向 stdin 和 stdout 是一种操作系统特性，而不是 C 语言本身的特性。但它确实从另一个方面说明了流的灵活性。有关重定向的更详细的信息，请参阅操作系统文档。

14.6　何时使用 fprintf()

前面指出过，fprintf()与 printf()相同，只是可以指定输出流。fprintf()主要用于处理磁盘文件，这将在第

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

16 天的课程中介绍。这里介绍该函数的其他两种用途。

14.6.1 使用 stderr

stderr（标准错误流）是预定义的流之一。程序的错误消息通常被发送到 stderr，而不是 stdout。原因何在呢？

您已知道，被发送到 stdout 的输出可被重定向到屏幕之外的其他地方。如果 stdout 被重定向，则用户可能无法看到程序发送到 stdout 的错误消息。不同于 stdout，stderr 不能被重定向，它总是与屏幕相连（至少在 DOS 操作系统中如此——UNIX 系统运行重定向 stderr）。通过将错误消息发送到 stderr，可以确保用户总是能看到。为此，可以使用 fprintf()：

```
fprintf(stderr, "An error has occurred.");
```

您可以编写一个处理错误消息的函数，并在发生错误时调用该函数，而不是调用 fprintf()：

```
error_message("An error has occurred.");

void error_message(char *msg)
{
    fprintf(stderr, msg);
}
```

使用自己定义的函数，而不是直接调用 fprintf()，这可以增加灵活性（这是结构化编程的优点之一）。例如，则某些情况下，您希望将程序的错误消息发送到打印机或磁盘文件。您只需修改 error_message()，便可以将输出发送到所需的目的地。

将输出发送到打印机

在 DOS 和 Windows 系统中，可以使用预定义的流 stdprn 将输出发送到打印机。在 IBM PC 或兼容机中，stdprn 流与设备 LPT1:（第一个并行打印机端口）相连。程序清单 14.15 是一个简单的范例。



注意：要使用 stdprn，必须关闭编译器中的 ANSI 兼容选项。更详细的信息，请参阅编译器的用户手册。由于 stdprn 并非 ANSI 标准的一部分，因此您的编译器可能不支持。

程序清单 14.15　　printer.c：将输出发送到打印机

```
1:  /* Demonstrates printer output. */
2:  /* stdprn is not ANSI standard  */
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:      float f = 2.0134;
8:
9:      fprintf(stdprn, "\nThis message is printed.\r\n");
10:     fprintf(stdprn, "And now some numbers:\r\n");
11:     fprintf(stdprn, "The square of %f is %f.", f, f*f);
12:
13:     /* Send a form feed. */
14:     fprintf(stdprn, "\f");
15:
16:     return 0;
17: }
```

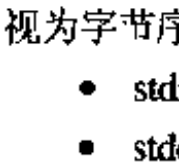

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

该程序的输出如下：

```
This message is printed.
And now some numbers:
The square of 2.013400 is 4.053780.
```



注意：上述输出由打印机打印，而不是显示在屏幕上。

如果您的计算机运行的是 DOS 操作系统，其 LPT1:端口连接了打印机，那么您可以编译并运行该程序。该程序在纸张上打印三行内容。第 14 行将“\f”发送给打印机。\f 是一个转义序列，表示换页，导致打印机在下一页进行打印（对于激光打印机，则弹出当前页）。

应　该	不应该
对于要将输出发送到 stdout、stderr、stdprn 或其他流的程序，一定要使用 fprintf()函数。 一定要使用 fprintf()和 stderr 将错误消息打印到屏幕上。 一定要创建 error_message 这样的函数来提高代码的结构化程度和可维护性。	不要对 stderr 进行重定向。 除了打印错误消息或警告外，不要使用 stderr。

14.7　总　结

今天的课程很长，介绍了大量关于程序输入/输出的信息。您学习了如何使用流——将所有的输入和输出视为字节序列。您还知道，ANSI C 有三个预定义的流：

- stdin：键盘；
- stdout：屏幕；
- stderr：屏幕。

键盘输入是通过 stdin 流进入的。使用 C 语言中的标准库函数，可以逐字符、逐行地读取键盘输入，也可以读取格式化数字和字符串。字符输入可能被缓冲、不被缓冲，也可能被回显、不被回显。

到屏幕的输出通常是通过 stdout 流进行的。与输入一样，也可以逐字符、逐行地显示程序输出，还可以以格式化数字或字符串的方式显示。要将输出发送到打印机，可以使用 fprintf()将数据发送给 stdprn 流。

使用 stdin 和 stdout 时，可以对输入和输出进行重定向。输入可以来自磁盘文件而不是键盘，而输出可以被发送到磁盘文件或打印机，而不是屏幕。

您还知道，为何应将错误消息发送给 stderr 流，而不是 stdout 流。由于 stderr 通常与屏幕相连，这样即使程序输出被重定向，用户也能看到错误消息。

另外，您还学习了其他两种不属于 ANSI 标准的流：

- stdprn：打印机；
- stdaux：通信端口。

14.8　问与答

问：如果试图从输出流中读取输入，情况将如何？

答：可以这样做，但程序将无法运行。例如，如果您试图在 fscanf()中使用 stdprn，程序将通过编译并生成可执行文件，但打印机无法发送输入，因此程序将无法正确地运行。

问：如果对标准流进行重定向，情况将如何？

答：这样做可能导致问题。如果对流进行重定向，则以后需要再次使用它时必须恢复。本章介绍的很多

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

函数都使用了标准流。它们使用的流是相同的，如果您在一个地方修改了流，则其他函数使用它时，它也被修改。例如，您可以在本章的某个程序清单中将 stdout 重定向到 stdprn，并看看发生的情况。

问：在程序中使用非 ANSI 函数有何危险？

答：大多数编译器都提供了很多有用的非 ANSI 标准函数。如果您打算一直使用同一个编译器，且无需将代码移植到其他编译器或平台中，则不会有任何问题。但如果您要使用其他的编译器或平台，则应该注意 ANSI 兼容性。

问：为何不能总是使用 fprintf()代替 printf()，使用 fscanf()代替 scanf()？

答：使用标准输入和输出流时，应使用 printf()和 scanf()；使用这些更简单的函数时，无需为其他流担心。

14.9 作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

14.9.1 小测验

1. 流是什么？C 程序使用流来干什么？
2. 下列设备分别是输入设备还是输出设备？
 a. 打印机；
 b. 键盘；
 c. 调制解调器；
 d. 显示器；
 e. 磁盘驱动器。
3. 指出 5 种预定义的流及其连接的设备。
4. 下列各函数分别使用哪个流？
 a. `printf()`
 b. `puts()`
 c. `scanf()`
 d. `gets()`
 e. `fprintf()`
5. 从 stdin 读取字符时，缓冲和不缓冲之间有何区别？
6. 从 stdin 读取字符时，回显和不回显之间有何区别？
7. 使用 ungetc()可以一次恢复多个字符吗？可以恢复 EOF 字符吗？
8. 使用行输入函数时，如何确定行尾？
9. 下面哪些是合法的类型说明符？
 a. `"%d"`
 b. `"%4d"`
 c. `"%3i%c"`
 d. `"%q%d"`
 e. `"%%%i"`
 f. `"%91d"`
10. stderr 和 stdout 之间有何区别？

14.9.2 练习

1. 编写一条语句，将“Hello World”打印到屏幕上。
2. 使用两个不同的函数来完成练习 1 中的任务。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

3. 编写一条语句，将“Hello Auxiliary Port”输出到标准辅助端口。

4. 编写一条语句，最多读取 30 个字符。遇到*后，截除余下的字符。

5. 编写一条打印下述内容的语句：

```
Jack asked, 'What is a backslash?"
Jill said, "It is '\'"
```

由于下面的练习有多种答案，因此附录 F 没有提供这些练习的答案，但您应该尝试完成它们。

6. 选做题：编写一个程序，将一个文件的内容逐字符地重定向到打印机。

7. 选做题：编写一个程序，通过重定向来从磁盘文件读取输入，计算每个字母在文件中出现的次数，并将其显示在屏幕上（附录 F 包含提示）。

8. 选做题：编写一个程序，打印源代码文件。通过重定向来输入源代码文件，并使用 fprintf()进行打印。

9. 选做题：修改练习 8 中编写的程序，以便打印程序清单时加上行号（附录 F 做了提示）。

10.选做题：编写一个“打字”程序，它读取键盘输入，将其回显到屏幕上，然后在打印机上打印它们。该程序应计算行数，以便在必要时换页。使用一个功能键来结束程序。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第二周复习

至此，您完成了学习C语言编程的第二周课程。现在，您应该熟悉C语言，因为您已经学习了所有的基本语句。下面的程序使用了本周介绍的很多知识。



注意：左边的编号指出了代码涉及的主题是在哪一天介绍的，如果您对某些代码不太明白，可参阅相应的课程，获取更详细的信息。

程序清单R2.1　　week2.c：第二周复习的程序清单

```
1:  /*------------------------------------------------- */
2:  /* Program Name:  week2.c                           */
3:  /* program to enter information for up to 100       */
4:  /* people.  The program prints a report             */
5:  /*          based on the numbers entered.           */
6:  /*------------------------------------------------- */
7:  /*----------------------*/
8:  /* included files       */
9:  /*----------------------*/
10: #include <stdio.h>
11: #include <stdlib.h>
12:
13: /*----------------------*/
14: /* defined constants    */
15: /*----------------------*/
16: #define MAX   100
17: #define YES   1
18: #define NO    0
19:
20: /*----------------------*/
21: /* variables            */
22: /*----------------------*/
23:
CH 11   24: struct record {
25:     char fname[15+1];          /* first name + NULL     */
CH 10   26:     char lname[20+1];          /* last name + NULL      */
27:     char phone[9+1];           /* phone number + NULL   */
28:     long income;               /* incomes               */
29:     int  month;                /* birthday month        */
30:     int  day;                  /* birthday day          */
31:     int  year;                 /* birthday year         */
```

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
CH 11    34:  struct record list[MAX];      /* declare actual structure */
CH 12    36:  int last_entry = 0;          /* total number of entries  */
CH 14    45:  void clear_kb(void);
CH 12    53:    int cont = YES;
CH 14    66:      ch = getchar();
CH 14    68:      fflush(stdin);  /* remove extra characters from keyboard buffer */
CH 13    70:      switch( ch )
CH 13    79:      default:  printf("\n\nInvalid choice, Please select 1 to 3!");
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
84:  }
85:  /*------ ---------------------------------------- *
86:   *  Function:  get_data()                         *
87:   *  Purpose: This function gets the data from the user. It   *
88:   *        continues to get data until either 100 people are  *
89:   *        entered, or the user chooses not to continue.      *
90:   *  Returns: nothing                                        *
91:   *Notes: This allows 0/0/0 to be entered for birthdates in   *
92:   *   case the user is unsure.  It also allows for 31 days    *
93:   *      in each month.                                       *
94:   *------- ---------------------------------------------- */
95:
96:  void get_data(void)
97:  {
CH 11  98:    int cont;
99:
100:   for ( cont = YES; last_entry < MAX && cont == YES;last_entry++ )
101:   {
102:     printf("\n\nEnter information for Person %d.",last_entry+1 );
103:
104:     printf("\n\nEnter first name: ");
CH 14  105:    gets(list[last_entry].fname);
106:
107:     printf("\nEnter last name: ");
CH 14  108:    gets(list[last_entry].lname);
109:
110:     printf("\nEnter phone in 123-4567 format: ");
CH 14  111:    gets(list[last_entry].phone);
112:
113:     printf("\nEnter Yearly Income (whole dollars): ");
CH  09 114:   scanf("%ld", &list[last_entry].income);
115:
116:     printf("\nEnter Birthday:");
117:
118:      do
119:      {
120:        printf("\n\tMonth (0 - 12): ");
CH 09  121:        scanf("%d", &list[last_entry].month);
122:     }while ( list[last_entry].month < 0 ||
123:              list[last_entry].month > 12 );
124:
125:     do
126:     {
127:        printf("\n\tDay (0 - 31): ");
CH 09  128:        scanf("%d", &list[last_entry].day);
129:     }while ( list[last_entry].day <  0 ||
130:              list[last_entry].day > 31 );
131:
132:     do
133:     {
134:        printf("\n\tYear (1800 - 1997): ");
CH 09  135:       scanf("%d", &list[last_entry].year);
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
136:      }while (list[last_entry].year != 0 &&
137:             (list[last_entry].year < 1800 ||
138:              list[last_entry].year > 1997 ));
139:
140:     cont = continue_function();
141:    }
142:
143:   if( last_entry == MAX)
144:     printf("\n\nMaximum Number of Names has been entered!\n");
145: }
146:
147: /*-----------------------------------------------------------*
148:  *  Function:  display_report()                              *
149:  * Purpose:    This function displays a report to the screen *
150:  *  Returns:   nothing                                       *
151:  *  Notes:     More information could be displayed.          *
152:  *             Change stdout to stdprn to Print report       *
153:  *-----------------------------------------------------------*/
154:
155: void display_report()
156: {
CH 12  157:    long   month_total = 0,
158:           grand_total = 0;      /* For totals */
CH 12  159:    int    x, y;
160:
CH 14  161:    fprintf(stdout, "\n\n");     /* skip a few lines */
162:    fprintf(stdout, "\n          REPORT");
163:    fprintf(stdout, "\n         ========");
164:
165:    for( x = 0; x <= 12; x++ )/* for each month, including 0 */
166:     {
167:       month_total = 0;
168:       for( y = 0; y < last_entry; y++ )
169:       {
170:          if( list[y].month == x )
171:          {
172:             fprintf(stdout,"\n\t%s %s %s %ld",list[y].fname,
CH 11  173:                    list[y].lname, list[y].phone,list[y].income);
174:             month_total += list[y].income;
175:          }
176:       }
CH 14  177: fprintf(stdout, "\nTotal for month %d is %ld",x,month_total);
178:       grand_total += month_total;
179:     }
180:    fprintf(stdout, "\n\nReport totals:");
181:    fprintf(stdout, "\nTotal Income is %ld", grand_total);
182:    fprintf(stdout, "\nAverage Income is %ld", grand_total/last_entry );
183:
184:    fprintf(stdout, "\n\n* * * End of Report * * **");
185: }
186:
187: /*-----------------------------------------------------------*
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
          188:  *  Function:  continue_function()                              *
          189:  *Purpose: This function asks the user if they wish to continue. *
          190:  *  Returns:YES - if user wishes to continue                     *
          191:  *          NO  - if user wishes to quit                         *
          192:  *-------------------------------------------------------------  */
          193:
          194: int continue_function( void )
          195: {
CH 13     196:    int ch;
          197:
          198:    printf("\n\nDo you wish to continue? (Y)es/(N)o: ");
          199:
CH 14     200:    fflush(stdin);
CH 14     201:    ch = getchar();
          202:
          203:    while( ch != 'n' && ch != 'N'&& ch != 'y'&& ch != 'y')
          204:    {|'N'|to Quit or|'y'|to Continue:"};
          205:       printf("\n%c is invalid!", ch);
          206:       printf("\n\nPlease enter |'N|'t0 Quit or |'Y|'to Continue");
          207:
CH 14     208:       fflush(stdin);     /* clear keyboard buffer (stdin) */
          209:       ch = getchar();
          210:    }
          211:
          212:
          213:   clear_kb();/* this function is similar to fflush(stdin) */
          214:
          215:   if|ch == 'n' || ch == 'N'}
          216:       return(NO);
          217:    else
          218:       return(YES);
          219: }
          220:
          221: /*-------------------------------------------------------------- *
          222:  *  Function:  clear_kb()                                        *
          223:  *Purpose:This function clears the keyboard of extra characters. *
          224:  *  Returns:   Nothing                                           *
          225:  *  Note: This function could be replaced by fflush(stdin);      *
          226:  *-------------------------------------------------------------  */
          227: void clear_kb(void)
          228: {
CH 09     229:     char junk[80];
CH 14     230:     gets(junk);
          231: }
```

分析：随着您越来越深入地学习 C 语言，程序也越来越长。虽然该程序是在第一周复习的程序清单的基础上修改而成的，但对其中的一些任务进行了修改，并添加了一些任务。与第一周复习一样，您可以最多输入 100 组数据，这些数据是关于人的信息。该程序能够在您输入数据的同时，显示报表；而第一周复习中的程序只能在输入完数据后，才能显示报表。

另外，添加了一个用于存储数据的结构。该结构是在第 24～32 行定义的。正如第 11 天的课程介绍的，结构通常用于将类似的数据分组。该程序将每个人的所有信息存储在一个名为 record 的结构中。其中的大部分数据与以前相同，但有一些数据项是新增的。第 25～27 行是三个字符数组（或字符串），用于存储姓、名

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

和电话号码。这些字符串的长度都被声明为比实际存储的最大字符数大 1。第 10 天的课程介绍过，多出的这个位置用于存储标识字符串末尾的空字符。

该程序演示了如何正确地使用变量作用域。第 34 和 36 行声明了两个全局变量。第 36 行使用 int 变量 last_entry 来存储输入了多少人的信息，这类似于第一周复习中使用的 ctr 变量。另一个全局变量是 list[MAX]，它是一个由 record 结构组成的数组。程序中的每个函数都使用了局部变量。其中要特别注意的是 display_report()函数中的 month_total、grand_total、x 和 y（第 157～159 行）。在第一周复习中，这些变量被声明为全局的。由于这些变量只用于 display_report()中，因此最好将他们声明为局部的。

第 70～81 行使用一个新的控制语句 switch。使用 switch 语句来代替多个 if...else 语句使代码更容易理解。第 72～79 行根据用户选择的菜单项执行不同的任务。其中还包含了 default 语句，以处理用户输入的不是有效的菜单项的情况。

对 get_data()函数做了一些修改。第 104 提示用户输入一个字符串。第 105 行使用 gets()（参见第 14 天的课程）来读取人名。gets()函数读取一个字符串，并将其存储到 list[last_entry].fname 中，这是将名存储到结构的成员 fname 中。

对 display_report()也做了修改。它使用 fprintf()而不是 printf()来显示信息。这样修改的原因很简单。如果要使用打印机打印报告，而不是将它显示到屏幕上，则只需将每个 fprintf()语句中的 stdout 修改为 stdprn 即可。有关 stdout、stdprn 和 fprintf()的内容，请参阅第 14 天的课程。stdout 和 stdprn 分别是与屏幕和打印机相连的流。

对 cuntinue_function()（第 194～219 行）也做了修改。现在，用户用 Y 或 N（而不是 0 或 1）来回答问题，这对用户更友好。另外，第 213 行使用程序清单 13.9 中的 clear_kb()来清除用户输入的多余字符。同时，还使用了 fflush()函数来清除缓冲区中的字符。



注意：可以在第 229 和 230 行中使用 fflush(stdin);，而不会对程序有任何影响。但您不能将所有的 fflush()都替换为 clear_kb()。如果您不明白其原因，请复习第 14 天的课程。

该程序使用了前两周介绍的知识。第二周介绍的一些概念使得程序的功能更强，代码编写起来更容易。第三周的课程将更深入地介绍这些概念。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

第三周课程

至此，您阅读完了学习如何使用C语言进行编程的前两周课程。您应该已熟悉了C语言，并接触到了该语言中最重要的领域。然而，需要学习的内容还很多，这一周的课程将进行介绍。

本周课程的内容

本周将完成对C语言的学习。本周将介绍一些新的主题，同时更深入地讨论前两周介绍过的一些主题。

阅读完本周的课程后，您将对C语言有一个全面的了解。第15天的课程讨论C语言中最难学也是功能最强大的部分——指针的高级用法。与第9天一样，您应额外花些时间来消化这些主题。第16天的课程将介绍对大多数应用程序来说都是必不可少的东西——磁盘文件。您将学习如何使用磁盘文件来存储和检索数据。第17天的课程将讨论所有用于处理文本的工具。第18天的课程将介绍有关函数的高级内容。第19天的课程介绍各种很有用的库函数的细节。第20天的课程将详细介绍内存管理。第21天的课程将介绍有关C语言的一些杂项信息，解释诸如命令行参数和预处理器编译指令等主题。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第 15 天课程　有关指针的高级主题

第 9 天的课程介绍了有关指针的基本知识，指针是 C 编程语言中至关重要的一部分。今天将更进一步介绍一些有关指针的高级主题，它们能够增加编程的灵活性。今天将介绍以下内容：

- 如何声明指向指针的指针？
- 如何将指针用于多维数组？
- 如何声明指针数组？
- 如何声明函数指针？
- 如何使用指针来创建链表，以存储数据？

15.1　声明指向指针的指针

第 9 天的课程介绍过，指针是一个数值变量，其值为另一个变量的地址。使用间接运算符（*）来声明指针。例如，下面的声明：

```
int *ptr;
```

声明了一个名为 ptr 的指针，该指针可以指向一个 int 变量。可以使用地址运算符（&）让指针指向某个相应类型的变量。假设 myVar 已经被声明为一个 int 变量，则下面的语句：

```
ptr = &myVar;
```

将变量 myVar 的地址赋给 ptr，使得 ptr 指向 myVar。同样，通过间接运算符，可以使用指针来存取被指向的变量。下面的两条语句都是将 12 赋给 myVar：

```
myVar = 12;
*ptr = 12;
```

由于指针本身是一个数值变量，它被存储在计算机内存的特定地址处。因此，可以创建指向指针的指针——一个变量，其值为一个指针的地址。下面是一个例子：

```
int myVar = 12;             /* myVar is a type int variable. */
int *ptr = &myVar;          /* ptr is a pointer to myVar. */
int **ptr_to_ptr = &ptr;    /* ptr_to_ptr is a pointer to a */
                            /* pointer to type int. */
```

声明指向指针的指针时，使用两个间接运算符（**）。同样，可以使用两个间接运算符来存取被指向的变量。因此，下面的语句：

```
**ptr_to_ptr =12;
```

将 12 赋给变量 myVar。而下面的语句：

```
printf("%d", **ptr_to_ptr);
```

将 myVar 的值显示到屏幕上。如果您错误地使用一个间接运算符，将发生错误。下面的语句：

```
*ptr_to_ptr =12;
```

将 12 赋给 ptr，导致 ptr 指向地址 12 处存储的值。这显然是不对的。

声明和使用指向指针的指针被称为多重间接（multiple indirection）。图 15.1 说明了变量、指针和指向指

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

针的指针之间的关系。对于多重间接的层数没有任何限制，但通常多于两层时没有任何好处，这样做无疑是自找麻烦。

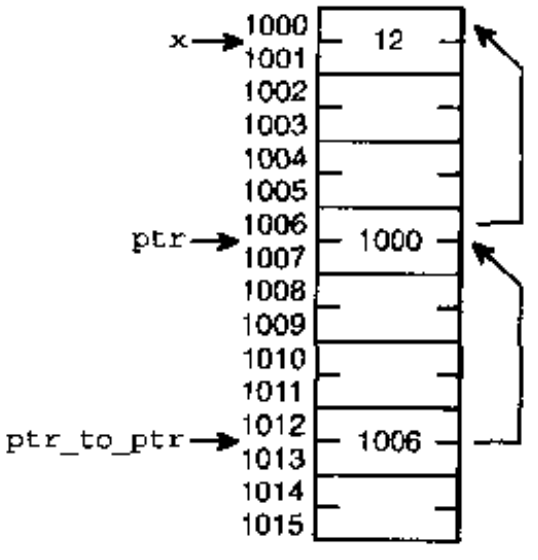


图 15.1　指向指针的指针

指向指针的指针有何用途呢？其最常见的用途涉及到指针数组（将在今天课程的后面介绍）。程序清单 19.5 是一个使用多重间接的范例。

15.2　指针和多维数组

第 8 天的课程介绍了指针和数组之间的特殊关系。具体地说，不带方括号的数组名是一个指针，它指向该数组的第一个元素。因此，存取数组时，使用指针表示法更容易。然而，前面的这些范例针对的是一维数组。对于多维数组，情况又如何呢？

声明多维数组时，使用多对方括号，每维一对。例如，下面的语句声明了一个二维数组，该数组包含 8 个 int 的变量：

```
int multi[2][4];
```

可以将二维数组看作是由行和列组成——这里为两行 4 列。然而，还有另一种看待多维数组的方式，这种方式与 C 语言实际处理数组的方式更吻合。可以将 multi 看作是一个包含两个元素的数组，而其中每个元素都是一个包含 4 个 int 变量的数组。

如果您对此还不清楚，请参考图 15.2，该图将上述数组声明语句分解为多个组成部分。

对该声明中各个组成部分的解释如下：

1. 声明了一个名为 multi 的数组；
2. 数组 multi 包含两个元素；
3. 其中每个元素都包含 4 个元素；
4. 4 个元素中的每一个的类型都是 int。

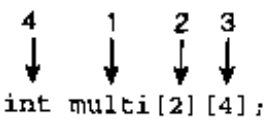


图 15.2　多维数组声明的组成部分

阅读多维数组的声明时，从数组名开始往右阅读，每次一对方括号。阅读到最后一对方括号后，跳转到声明的开始位置，以确定数组的基本数据类型。

采用数组的数组方式时，可以将多维数组看作是如图 15.3 所示。

现在回到数组名作为指针的主题（毕竟今天的课程是介绍指针的）。与一维数组一样，多维数组的名称也是一个指向第一个元素的指针。继续前面的例子，multi 是一个指针，它指向被声明为 int multi[2][4]的数组的

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

第一个元素。multi 数组的第一个元素是什么呢？不是 int 变量 multi[0][0]。multi 是一个由数组组成的数组，因此它的第一个元素为 multi[0]——一个由 4 个 int 变量组成的数组（multi 包含两个这样的数组）。

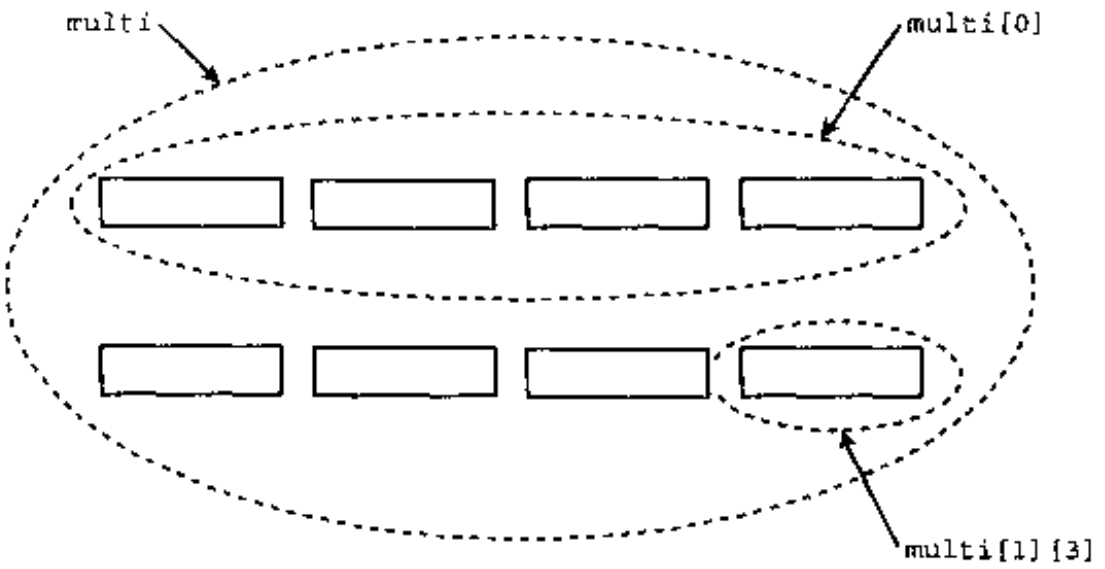


图 15.3　二维数组可被视为一个由数组组成的数组

既然 multi[0]也是一个数组，它是否也指向某种东西呢？是的，确实如此。multi[0]指向它的第一个元素——multi[0][0]。您可能会问，为何 multi[0]是一个指针呢？

不带方括号的数组名是一个指针，它指向数组的第一个元素。multi[0]是数组 multi[0][0]的名称（不带最后一对方括号），因此它是一个指针。

如果您感到困惑，请不用担心。这些内容比较难掌握。对于任何 n 维数组，存在下述规则，记住这些规则会对您有所帮助：

- 数组名后带 n 对方括号（当然，其中每对方括号中都包含一个合适的索引）时，为数组的数据（即存储在指定数组元素中的数据）；
- 数组名后带的方括号少于 n 对时，为一个指向数组元素的指针。

因此，在这个例子中，multi 是一个指针，multi[0]也是一个指针，而 multi[0][0]是数组的数据。

下面来看看这些指针指向的是什么。程序清单 15.1 声明了一个二维数组——类似于前面的范例中使用的，然后打印指针的值。该程序还打印了第一个数组元素的地址。

程序清单 15.1　　multi.c：多维数组和指针之间的关系

```
1:   /* Demonstrates pointers and multidimensional arrays. */
2:
3:   #include <stdio.h>
4:
5:   int multi[2][4];
6:
7:   int main( void )
8:   {
9:       printf("\nmulti = %u", multi);
10:      printf("\nmulti[0] = %u", multi[0]);
11:      printf("\n&multi[0][0] = %u\n", &multi[0][0]);
12:
13:      return 0;
14:  }
```

该程序的输出如下：

```
multi = 4206592
multi[0] = 4206592
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
&multi[0][0] = 4206592
```

您在运行该程序时，值可能不是 4206592，但三个值应该相同。数组 multi 的地址与数组 multi[0]的地址相同，它们等于数组中第一个整数（multi[0][0]）的地址。

既然这三个指针的值都相同，那么就这个程序而言，它们之间有何区别呢？第 9 天的课程介绍过，C 编译器知道指针指向的是什么。更准确地说，编译器知道指针指向的元素的长度。

前面使用的各个元素的长度分别是多少呢？程序清单 15.2 使用运算符 sizeof()来显示这些元素的长度（单位为字节）。

程序清单 15.2　　multisize.c：确定元素的长度

```
1:  /* Demonstrates the sizes of multidimensiona_ array elements. */
2:
3:  #include <stdio.h>
4:
5:  int multi[2][4];
6:
7:  int main( void )
8:  {
9:     printf("\nThe size of multi = %u", sizeof(multi));
10:    printf("\nThe size of multi[0] = %u", sizeof(multi[0]));
11:    printf("\nThe size of multi[0][0] = %u\n", sizeof(multi[0][0]));
12:
13:    return 0;
14: }
```

该程序的输出如下（假设您的编译器使用 4 个字节的内存来存储整数）：

```
The size of multi = 32
The size of multi[0] = 16
The size of multi[0][0] = 4
```

注意，在不同机器上运行该程序时，显示的长度值可能不同。

分析：如果在 32 位的操作系统上运行该程序，则输出很可能是 32、16 和 4。这是因为在这样的系统中，int 变量占用 4 个字节的内存。对于老式的 16 位系统，输出可能为 16、8、2。

来看看这些长度值。数组 multi 包含两个数组，其中每个数组都包含 4 个 int 变量。每个 int 变量占用 4 个字节的内存。由于总共有 8 个 int 变量，因此数组的长度为 32 个字节。

multi[0]是一个包含四个 int 变量的数组。其中每个 int 变量占用 4 个字节，因此 multi[0]的长度为 16 个字节。

最后，multi[0][0]是一个 int 变量，因此其长度为 4 个字节。

现在记住这些长度值，并回忆一下第 9 天课程对指针算术的讨论。C 编译器知道被指向的对象的长度，而指针算术则考虑到了这种长度。当您对指针执行递增运算时，其值将增加它指向的对象长度，从而使之指向下一个对象。

将这种规则应用到这个例子时，multi 是一个指向数组的指针，该数组包含 4 个 int 元素，长度为 16。如果对 multi 执行递增运算，则其值将增加 16（包含 4 个 int 元素的数组的长度）。如果 multi 指向 multi[0]，则 mullti+1 指向的是 multi[1]。程序清单 15.3 检测了这个理论。

程序清单 15.3　　multimath.c：将指针算术用于多维数组

```
1:  /*  Demonstrates pointer arithmetic with pointers */
2:  /*  to multidimensional arrays. */
3:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
4:  #include <stdio.h>
5:
6:  int multi[2][4];
7:
8:  int main( void )
9:  {
10:    printf("\nThe value of (multi) = %u", multi);
11:    printf("\nThe value of (multi + 1) = %u", (multi+1));
12:    printf("\nThe address of multi[1] = %u\n", &multi[1]);
13:
14:    return 0;
15: }
```

该程序的输出如下：

```
The value of (multi) = 4206592
The value of (multi + 1) = 4206608
The address of multi[1] = 4206608
```

分析：您运行该程序时，显示的值可能不同，但这些值之间的关系是相同的。multi+1 的值比 multi 大 16（在老式的 16 位系统中大 8），它指向数组的下一个元素：multi[1]。

从这个例子可知，multi 是一个指针，它指向 multi[0]。另外，mulit[0]本身也是一个指针，它指向 multi[0][0]。因此 multi 是一个指向指针的指针。要通过 multi 来存取数组的数据，必须使用两个间接运算符。要打印 multi[0][0]的值，可以使用下述三条语句中的任何一条：

```
printf("%d", multi[0][0]);
printf("%d", *multi[0]);
printf("%d", **multi);
```

这些概念同样适用于三维或三维以上的数组。因此三维数组是一个包含数组的数组，其中的每个数组又都是一个二维数组，而二维数组中的每个元素又是一个一维数组。

上述有关多维数组和指针的内容可能比较难理解。使用多维数组时，请记住这样一点：n 维数组的元素是 n-1 维数组。当 n 为 1 时，元素为数组声明的开头指定的类型的变量。

至此，您使用的是数组名。数组名是指针常量，无法修改。如何声明一个指向多维数组的元素的指针变量呢？在前面的例子中，声明了一个二维数组：

```
int multi[2][4];
```

要声明一个名为 ptr，且能够指向 multi 的元素（即能够指向包含 4 个 int 变量的数组）的指针变量，可以这样做：

```
int (*ptr)[4];
```

然后，用下面的语句使 ptr 指向 multi 的第一个元素：

```
ptr = multi;
```

您可能会问，为何前面声明指针时需要使用圆括号？方括号的优先级高于*。如果这样编写代码：

```
int *ptr[4];
```

则声明的是一个数组，该数组包含 4 个 int 指针。实际上，您可以声明并使用指针数组，但您现在要做的并非如此。

如何使用指向多维数组的元素的指针呢？与一维数组一样，要将多维数组传递给函数，也必须使用指针。程序清单 15.4 演示了这一点，该程序使用了两种方法将多维数组传递给函数。

程序清单 15.4　　ptrmulti.c：使用指针来将维数组传递给函数

```
1:  /* Demonstrates passing a pointer to a multidimensional */
2:  /* array to a function. */
3:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
4:  #include <stdio.h>
5:
6:  void printarray_1(int (*ptr)[4]);
7:  void printarray_2(int (*ptr)[4], int n);
8:
9:  int main( void )
10: {
11:    int  multi[3][4] = { { 1, 2, 3, 4 },
12:                         { 5, 6, 7, 8 },
13:                         { 9, 10, 11, 12 } };
14:
15:    /* ptr is a pointer to an array of 4 ints. */
16:
17:    int (*ptr)[4], count;
18:
19:    /* Set ptr to point to the first element of multi. */
20:
21:    ptr = multi;
22:
23:    /* With each loop, ptr is incremented to point to the next */
24:    /* element (that is, the next 4-element integer array) of multi. */
25:
26:    for (count = 0; count < 3; count++)
27:       printarray_1(ptr++);
28:
29:    puts("\n\nPress Enter...");
30:    getchar();
31:    printarray_2(multi, 3);
32:    printf("\n");
33:    return 0;
34: }
35:
36: void printarray_1(int (*ptr)[4])
37: {
38: /* Prints the elements of a single four-element integer array. */
39: /* p is a pointer to type int. You must use a type cast */
40: /* to make p equal to the address in ptr. */
41:
42:     int *p, count;
43:     p = (int *)ptr;
44:
45:     for (count = 0; count < 4; count++)
46:        printf("\n%d", *p++);
47: }
48:
49: void printarray_2(int (*ptr)[4], int n)
50: {
51: /* Prints the elements of an n by four-element integer array. */
52:
53:     int *p, count;
54:     p = (int *)ptr;
55:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
56:     for (count = 0; count < (4 * n); count++)
57:         printf("\n%d", *p--);
58: }
```

该程序的输出如下：

```
1
2
3
4
5
6
7
8
9
10
11
12

Press Enter...


1
2
3
4
5
6
7
8
9
10
11
12
```

分析：第 11～13 行声明并初始化了一个 int 数组——multi[3][4]。第 6 和 7 行是函数 printarray_1()和 printarray_2()的原型，这两个函数打印数组的内容。

函数 printarray_1()（第 36～47 行）只接受一个参数——一个指针，该指针指向一个包含 4 个 int 元素的数组。该函数打印数组的 4 个元素。main()首次在第 27 行调用函数 printarray_1()，它将指向 multi 的第一个元素（第一个包含 4 个 int 元素的数组）的指针传递给函数。然后，再调用该函数两次，每次调用时，将指针递增，使之指向 multi 数组的第二个元素和第三个元素。完成上述三次调用后，multi 的 12 个 int 元素便被显示在屏幕上。

第二个函数 printarray_2()采用了另一种方法。它也接受一个指针，该指针指向一个包含 4 个 int 元素的数组；但还接受一个 int 参数，该参数指出了多维数组包含的元素数目。这样，只需调用 printarray_2()一次（第 31 行），便可以显示 multi 的所有元素。

这两个函数都使用指针表示法来遍历数组中的各个 int 元素。这两个函数都使用了(int *)ptr（第 43 和 54 行），您可能不清楚其含义。(int *)用于强制类型转换，暂时将变量 ptr 的数据类型从声明的类型转换为新的类型。将 ptr 的值赋给 p 时，必须执行强制类型转换，因为它们是指向不同数据类型的指针（p 是一个 int 指针，而 ptr 是一个指向包含 4 个 int 元素的数组的指针）。C 语言不允许将一种指针的值赋给另一种指针。强制类型转换告诉编译器："在这条语句中，将 ptr 看作一个 int 指针。"第 20 天的课程将更详细地介绍强制类型转换。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

应　该	不应该
声明指向指针的指针时，一定要使用两个间接运算符（**）。 对指针执行递增运算时，增加的值为指针指向的类型的长度。	声明数组指针时，别忘了使用圆括号。 声明指向字符数组的指针的格式如下： char (*letters)[26]; 声明 char 指针数组的格式如下： char *letters[26];

15.3　指针数组

第 8 天的课程介绍过，数组是一组数据存储单元，它们的数据类型相同，并通过同一个名称来引用它们。由于指针也是一种数据类型，因此可以声明并使用指针数组。在有些情况下，指针数组的功能极其强大。

指针数组最常见的用途是用于处理字符串。第 10 天的课程介绍过，字符串是一个存储在内存中的字符序列，其开始位置由指向第一个字符的指针（char 指针）标识，末尾则通过空字符标记。声明并初始化一个 char 指针数组后，可以使用它来存取和操纵大量的字符串。数组中的每个元素都指向一个不同的字符串，通过遍历该数组，可以依次存取每个字符串。

15.3.1　复习字符串和指针

下面来复习一下第 10 天课程中有关字符串分配和初始化的内容。给字符串分配空间并对其进行初始化的方式之一是，声明一个 char 数组，如下所示：

```
char message[] = "This is the message.";
```

也可以声明一个 char 指针来完成这样的任务：

```
char *message = "This is the message.";
```

上述两个声明是等价的。无论采用哪种方式，编译器都将分配足够的空间来存储字符串及末尾的空字符。message 是一个指针，指向字符串的开始位置。下面两个声明的情况如何呢？

```
char message1[20];
char *message2;
```

第一个声明了一个长度为 20 个字符的 char 数组，message1 是一个指向数组开始位置的指针。虽然为数组分配了空间，但并没有对数组进行初始化，因此数组的内容是不确定的。第二行将 message2 声明为一个 char 指针。该语句并没有分配用于存储字符串的空间，而只分配了用于存储指针的空间。如果您要创建一个字符串，并让 message2 指向它，则必须首先为字符串分配空间。第 10 天的课程介绍了如何使用内存分配函数 malloc()来分配空间。请记住，对于任何字符串，都必须分配用于存储它的内存空间——这可以在编译时通过声明来分配，也可以在运行时通过 malloc()（或相关的内存分配函数）来分配。

15.3.2　声明 char 类型指针数组

如何声明指针数组呢？下面的语句声明了一个包含 10 个元素的 char 指针数组：

```
char *message[10];
```

数组 message[]的每个元素都是一个 char 指针。您可能猜到了，可以在声明的同时，对数组进行初始化，从而为字符串分配空间：

```
char *message[10] = { "one", "two", "three" };
```

上述声明的功能如下：

- 为一个包含 10 个元素的、名为 message 的数组分配空间，其中每个元素都是 char 指针；
- 在内存的某个地方（您无需关心到底在哪里）分配空间，用于存储三个初始字符串，其中每个字符串都以空字符结尾；
- 将 message[0]、message[1]和 message[2]分别初始化为指向第一个字符串（one）、第二个字符串（two）和第三个字符串（three）的第一个字符。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

图 15.4 对此做了说明，它指出了指针数组和字符串之间的关系。在这个例子中，没有对 message[3]至 message[9]进行初始化，使之指向某个字符串。

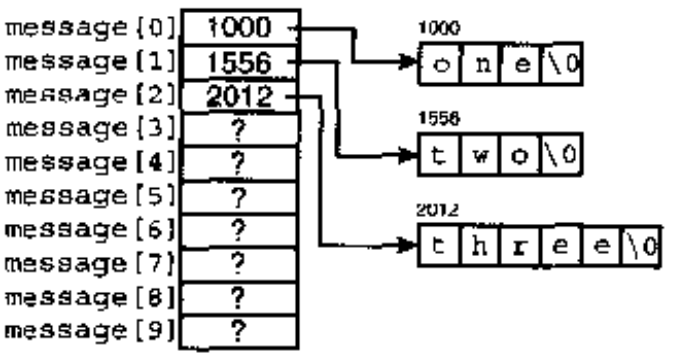


图 15.4 char 指针数组

程序清单 15.5 是一个使用指针数组的范例。

程序清单 15.5　　message.c：初始化并使用 char 指针数组

```
1:  /* Initializing an array of pointers to type char. */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:     char *message[8] = { "Four", "score", "and", "seven",
8:                          "years", "ago,", "our", "forefathers" };
9:     int count;
10:
11:    for (count = 0; count < 8; count++)
12:       printf("%s ", message[count]);
13:    printf("\n");
14:
15:    return 0;
16: }
```

该程序的输出如下：

```
Four score and seven years ago, our forefathers
```

分析：该程序声明了一个包含 8 个元素的 char 指针数组，并将这些元素初始化为指向 8 个字符串（第 7 和 8 行）。然后使用 for 循环将每个元素显示到屏幕上（第 11 和 12 行）。

您可能已经看出来了，操纵指针数组比操纵字符串本身容易得多。在更为复杂的程序中（如本章后面的一个程序），这种优势将是显而易见的。到时您将发现，使用函数时，这种优势将发挥到极至。与将多个字符串传递给函数相比，传递指针数组要容易得多。可以重写程序清单 15.5，使之使用一个函数来显示字符串，以说明这一点。修改后的程序如程序清单 15.6 所示。

程序清单 15.6　　message.c：将指针数组传递给函数

```
1:  /* Passing an array of pointers to a function. */
2:
3:  #include <stdio.h>
4:
5:  void print_strings(char *p[], int n);
6:
7:  int main( void )
8:  {
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
 9:     char *message[8] = { "Four", "score", "and", "seven",
10:                         "years", "ago,", "our", "forefathers" };
11:
12:     print_strings(message, 8);
13:     return 0;
14: }
15:
16: void print_strings(char *p[], int n)
17: {
18:     int count;
19:
20:     for (count = 0; count < n; count++)
21:        printf("%s ", p[count]);
22:     printf("\n");
23: }
```

该程序的输出如下：

```
Four score and seven years ago, our forefathers
```

从第 16 行可知，函数 print_string()接受两个参数：一个是 char 指针数组，另一个是数组的元素数目。因此，print_string()可用于打印任何指针数组指向的字符串。

您可能还记得，关于指向指针的指针一节指出过，后面将对其进行演示。现在就进行演示。程序清单 15.6 声明了一个指针数组，数组名是一个指针，它指向数组的第一个元素。将该数组传递给函数时，您实际上是传递了一个指向指针（第一个数组元素）的指针（数组名）。

15.3.3　范例

下面介绍一个更复杂的例子。程序清单 15.7 使用了前面介绍的很多编程技巧，其中包括指针数组。该程序从键盘读取多行输入，在读取时为每一行分配内存空间，并使用 char 指针数组来跟踪这些内容。当用户输入一个空行，指出结束输入后，程序按字母顺序对这些字符串进行排序，并将它们显示到屏幕上。

如果是从开零开始编写该程序，则应从结构化编程的角度来设计它。首先列出程序需要完成的任务：

1. 每次从键盘读取一行，直到用户输入空行为止；
2. 按字母顺序对各行文本进行排序；
3. 将排序后的各行文本显示到屏幕上。

上述任务列表表明，程序至少应包含三个函数，分别用于读取输入、对文本行进行排序以及显示文本行。输入函数（名为 get_lines()）应具备哪些功能呢？同样，可以制作一个清单：

1. 跟踪用户输入了多少行，并在用户输入完毕后将这个值返回给调用程序；
2. 不允许输入的行数超过指定的最大值；
3. 为每一行文本分配存储空间；
4. 将指向字符串的指针存储到数组中，以跟踪输入的所有文本行；
5. 当用户输入空行后，返回到调用程序。

下面来看看第二个函数——对文本行进行排序的函数。可以将它命名为 sort()。这里使用的排序技术很简单，对相邻的字符串进行比较，如果第二个字符串小于第一个，则交换它们的位置。更准确地说，函数对指针数组中相邻的指针指向的字符串进行比较，必要时交换两个指针的位置。

为确保排序的完整性，必须从头到尾遍历数组，依次将每对字符串进行比较，并在必要时交换它们的位置。对于一个包含 n 个元素的数组，必须遍历 n-1 次。为何必须这样？

每遍历数组一次，每个元素最多被移动一个位置。例如，如果本应排在第一的字符串位于最后，则第一次遍历数组时，它将被移到倒数第二；第二次遍历数组时，它将向前移动一个位置；依此类推。因此需要遍历 n-1 次才能将它移到最前面。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

这是一种低效而又笨拙的排序方法。然而，它易于实现和理解，对于该程序中对很少的字符串进行排序而言，这绰绰有余。

最后一个函数将排序后的文本行显示到屏幕上。实际上，程序清单 15.6 已经包含这样的函数，只需对其进行细微的修改，便可用于程序清单 15.7 中。

程序清单 15.7　　sort.c：从键盘读取文本行，对它们进行排序，然后将排序后的文本行显示到屏幕上

```
1:  /* Inputs a list of strings from the keyboard, scrts them, */
2:  /* and then displays them on the screen. */
3:  #include <stdlib.h>
4:  #include <stdio.h>
5:  #include <string.h>
6:
7:  #define MAXLINES 25
8:
9:  int get_lines(char *lines[]);
10: void sort(char *p[], int n);
11: void print_strings(char *p[], int n);
12:
13: char *lines[MAXLINES];
14:
15: int main( void )
16: {
17:    int number_of_lines;
18:
19:    /* Read in the lines from the keyboard. */
20:
21:    number_of_lines = get_lines(lines);
22:
23:    if ( number_of_lines < 0 )
24:    {
25:       puts(" Memory allocation error");
26:       exit(-1);
27:    }
28:
29:    sort(lines, number_of_lines);
30:    print_strings(lines, number_of_lines);
31:    return 0;
32: }
33:
34: int get_lincs(char *lines[])
35: {
36:    int n = 0;
37:    char buffer[80];  /* Temporary storage for each line. */
38:
39:    puts("Enter one line at time; enter a blank when done.");
40:
41:    while ((n < MAXLINES) && (gets(buffer) != 0) &&
42:          (buffer[0] != '\0'))
43:    {
44:       if ((lines[n] = (char *)malloc(strlen(buffer)+1)) == NULL)
45:          return -1;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
46:         strcpy( lines[n++], buffer );
47:     }
48:     return n;
49:
50: } /* End of get_lines() */
51:
52: void sort(char *p[], int n)
53: {
54:     int a, b;
55:     char *tmp;
56:
57:     for (a = 1; a < n; a++)
58:     {
59:         for (b = 0; b < n-1; b++)
60:         {
61:             if (strcmp(p[b], p[b+1]) > 0)
62:             {
63:                 tmp = p[b];
64:                 p[b] = p[b+1];
65:                 p[b+1] = tmp;
66:             }
67:         }
68:     }
69: }
70:
71: void print_strings(char *p[], int n)
72: {
73:     int count;
74:
75:     for (count = 0; count < n; count++)
76:         printf("%s\n", p[count]);
77: }
```

该程序的运行情况如下：

```
Enter one line at time; enter a blank when done.
dog
apple
zoo
program
merry
apple
dog
merry
program
zoo
```

分析：有必要介绍一下该程序的一些细节。使用了多个新的库函数来完成各种字符串操纵，这里对它们做一简要的介绍，更详细的细节请参阅第 17 天的课程。要使用这些函数，程序必须包含头文件 string.h。

在 get_lines()函数中，使用 while 语句（第 41 和 42 行）来控制输入。该语句如下（这里将它压缩成了一行）：

```
while ((n < MAXLINES) && (gets(buffer) != 0) && (buffer[0] != '\0'))
```

while 语句测试的条件由三部分组成。第一部分 n < MAXLINES 确保输入的行数不会超过指定的最大数

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

目；第二部分 gets(buffer) != 0 调用库函数 gets()从键盘读取一行到 buffer 中，并检查是否达到文件末尾或发生了其他错误；第三部分 buffer[0] != '\0'验证用户输入的一行的第一个字符是否为空字符，如果是，则说明输入的是空行。

如果上述三个条件中的任何一个不满足，则 while 语句将终止，然后返回调用程序，并将输入的函数作为返回值。如果所有三个条件都满足，则执行下面的 if 语句（第 44 行）：

```
if ((lines[n] = (char *)malloc(strlen(buffer)+1)) == NULL)
```

语句调用 malloc()来为刚才输入的字符串分配空间。strlen()函数返回传递给它的字符串的长度，返回的值被加 1，以便 malloc()分配的空间足以存储整个字符串和结尾的空字符。malloc()前面的(char *)（第 44 行）用于将 malloc()返回的指针强制转换为 char 指针。有关强制类型转换，将在第 20 天的课程中做更详细的介绍。

您可能还记得，库函数 malloc()返回一个指针。上述语句将 malloc()返回的指针赋给指针数组中的相应元素。如果 malloc()返回的是 NULL，则 if 循环返回到调用程序，返回值为-1。main()函数中的代码检查 get_lines()返回的值是否小于 0，如果是，则报告内存分配错误，然后程序终止（第 23～27 行）。

如果成功地分配了内存，则程序使用 strcpy()函数（第 46 行）将字符串从临时存储位置 buffer 处复制到 malloc()分配的存储空间中。然后，while 循环继续进行，读取另一行输入。

get_lines()执行完毕，并返回到 main()后，以下工作便完成了：

- 从键盘读取了一定行数的文本，并将它们作为以空字符结尾的字符串存储到了内存中；
- 数组 lines[]中包含了指向这些字符串的指针。这些指针在数组中的排列顺序与字符串的输入顺序相同。
- 变量 number_of_lines 的值为输入的行数。

现在可以进行排序了。您并不移动字符串，而只是移动数组 lines[]中的指针。来看看函数 sort()的代码。它将一个 for 循环嵌套在另一个 for 循环中（第 57～68 行）。外面的循环执行 number_of_lines-1 次。外部循环每次执行时，内部循环遍历指针数组，对字符串 n 和 n+1（n 为 0 到 numuber_of_lines-1）进行比较。比较是通过库函数 strcmp()（第 61 行）完成的，该函数接受两个指向字符串的指针。函数 strcmp()的返回值如下：

- 如果第一个字符串大于第二个，则返回一个大于 0 的值；
- 如果两个字符串相同，则返回 0；
- 如果第一个字符串小于第二个，则返回一个小于 0 的值。

在该程序中，如果 strcmp()返回的值大于 0，则说明第一个字符串"大于"第二个，必须交换它们的位置（即交换 lines[]中指向它们的指针的位置）。这是通过使用一个临时变量 tmp 来完成的（第 63～65 行）。

sort()函数执行完毕后，lines[]中的指针将被正确排列：指向“最小”的字符串的指针位于 lines[0]中，指向“次小”的字符串的指针位于 lines[1]中，依此类推。例如，假设用户依次输入了下列 5 行内容：

```
dog
apple
zoo
program
merry
```

则调用 sort()函数之前的情况如图 15.5 所示；而 sort()函数执行完毕后的情况如图 15.6 所示。

最后，程序调用函数 print_strings()将排序后的字符串显示到屏幕上。您应该熟悉这个函数，因为它与前一个程序清单中使用的一个函数类似。

程序清单 15.7 中的程序是到目前为止，本书中最为复杂的程序。它使用了以前的课程中介绍的很多编程技巧。有了上面的解释后，您应该能够理解该程序的工作原理以及每个步骤的细节。如果有什么地方不明白，请复习本书相关章节的内容，直到完全弄懂为止。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

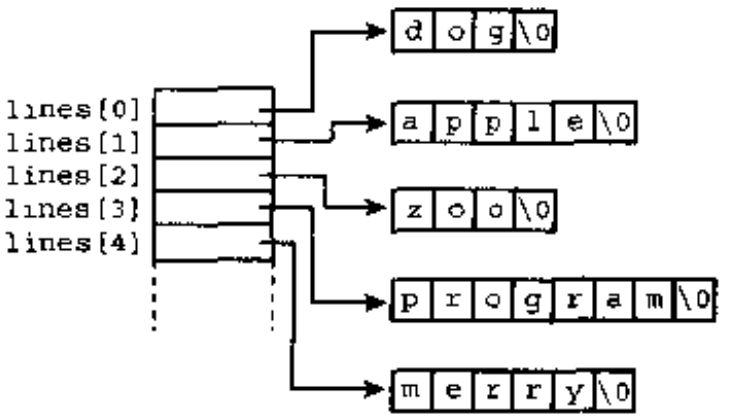


图 15.5　排序前，指针的排列顺序与字符串的输入顺序相同

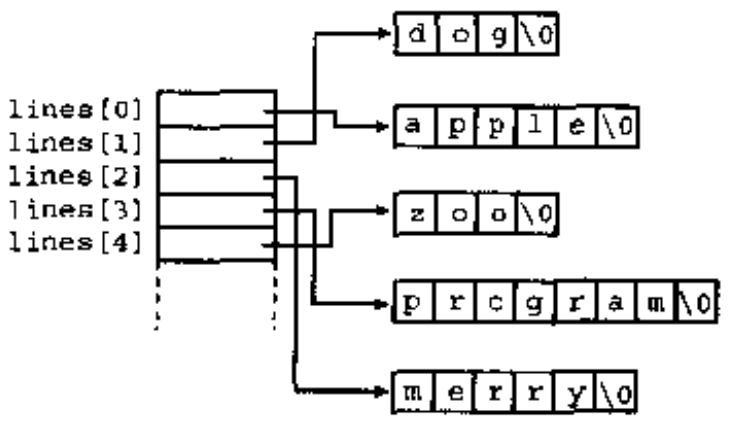


图 15.6　排序后，指针按字符串的字母顺序排列

15.4　函数指针

函数指针提供了另一种调用函数的方式。您可能会问，如何让指针指向函数呢？指针不是存储的是变量的地址吗？

指针存储的确实是地址，但不一定非得是变量的地址。程序运行时，每个函数的代码被存储在特定地址指定的内存中。函数指针存储的是函数的开始地址（入口）。

为何要使用函数指针呢？正如前面指出的，它提供了一种更灵活的调用函数的方式。它让程序能够从多个函数中选择一个对当前情况而言合适的函数。

15.4.1　声明函数指针

与所有的变量一样，使用函数之前也必须声明它。其声明的通用格式如下：

```
type (*ptr_to_func)(parameter_list);
```

该语句将 ptr_to_func 声明为指向一个函数的指针，使函数的返回类型为 type，并接受 parameter_list 指定的参数。下面是一些具体的例子：

```
int (*func1)(int x);
void (*func2)(double y, double z);
char (*func3)(char *p[]);
void (*func4)();
```

其中第一行将 func1 声明为指向某个函数的指针，该函数接受一个 int 参数，并返回一个 int 值。第二行将 func2 声明为一个指向某个函数的指针，该函数接受两个 double 参数，且返回类型为 void（没有返回值）。第 3 行将 func3 声明为指向某个函数的指针，该函数接受一个 char 指针数组作为参数，返回类型为 char。最后一行将 func4 声明为指向某个函数的指针，该函数不接受任何参数，且返回类型为 void。

为何要使用圆括号将指针名括起呢？对于第一个例子，为何不可以这样编写代码呢？

```
int *func1(int x);
```

原因在于间接运算符(*)的优先级低于将参数列表括起的圆括号。上述删除第一对圆括号的声明将 func1

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

声明为一个返回类型为 int 指针的函数（返回指针的函数将在第 18 天的课程中介绍）。声明函数指针时，一定要用圆括号将指针名和间接运算符括起，否则会出现问题。

15.4.2 初始化并使用函数指针

函数指针不但需要声明，还必须被初始化为指向某个函数。对被指向的函数的唯一要求是，其返回类型和参数列表必须与指针声明中的返回类型和参数列表匹配。例如，下面的代码声明并定义了一个函数以及一个指向该函数的指针：

```
float square(float x);     // The function prototype.
float (*ptr)(float x);     // The pointer declaration.
float square(float x)      // The function definition.
{
    return x * x;
}
```

由于函数 square()的参数和返回类型与指针 ptr 相同，因此可以将 ptr 初始化为指向 square，如下所示：

```
ptr = square;
```

然后，便可以使用指针来调用该函数，如下所示：

```
answer = ptr(x);
```

这很简单。程序清单 15.8 是一个真正的例子，它声明并初始化一个函数指针，然后调用函数两次：第一次使用函数名，而第二次使用指针。这两次调用的结果相同。

程序清单 15.8　　ptr.c：使用函数指针调用函数

```
1:  /* Demonstration of declaring and using a pointer to a function.*/
2:
3:  #include <stdio.h>
4:
5:  /* The function prototype. */
6:
7:  double square(double x);
8:
9:  /* The pointer declaration. */
10:
11: double (*ptr)(double x);
12:
13: int main( void )
14: {
15:    /* Initialize p to point to square(). */
16:
17:     ptr = square;
18:
19:    /* Call square() two ways. */
20:    printf("%f  %f\n", square(6.6), ptr(6.6));
21:    return 0;
22: }
23:
24: double square(double x)
25: {
26:    return x * x;
27: }
```

该程序的输出如下：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
43.560000  43.560000
```



> 注意：由于精度的影响，显示有些数字时，结果可能与正确值不完全相同。例如，43.56 可能被显示为 43.559999。

第 7 行声明了函数 square()，而第 11 行将 ptr 声明为指向某个函数的指针，该函数接受一个 double 参数，并返回一个 double 值，这与 square()的声明匹配。第 17 行将指针 ptr 的值设置为 square，注意，这里没有对 ptr 和 square 使用圆括号。第 20 行打印调用 square()和 ptr()的返回值。

不带圆括号的函数名是一个指向该函数的指针（与数组有些类似，不是吗？）。声明并使用一个独立的函数指针有何意义呢？函数名本身是一个指针常量，不可修改（这也类似于数组）。而指针变量可以修改。具体地说，可以根据需要让它指向不同的函数。

程序清单 15.9 调用了一个函数，并将一个 int 参数传递给它。函数根据该参数的值，将一个指针初始化为指向另外三个函数中的一个，然后使用该指针来调用相应的函数。这三个函数都将特定的消息显示到屏幕上。

程序清单 15.9　　ptr2.c：使用函数指针来根据情况调用不同的函数

```
1:  /* Using a pointer to call different functions. */
2:
3:  #include <stdio.h>
4:
5:  /* The function prototypes. */
6:
7:  void func1(int x);
8:  void one(void);
9:  void two(void);
10: void other(void);
11:
12:  int main( void )
13: {
14:     int nbr;
15:
16:     for (;;)
17:     {
18:        puts("\nEnter an integer between 1 and 10, 0 to exit: ");
19:        scanf("%d", &nbr);
20:
21:        if (nbr == 0)
22:           break;
23:        func1(nbr);
24:     }
25:     return 0;
26: }
27:
28: void func1(int val)
29: {
30:     /* The pointer to function. */
31:
32:     void (*ptr)(void);
33:
34:     if (val == 1)
35:        ptr = one;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
36:     else if (val == 2)
37:         ptr = two;
38:     else
39:         ptr = other;
40:
41:     ptr();
42: }
43:
44: void one(void)
45: {
46:     puts("You entered 1.");
47: }
48:
49: void two(void)
50: {
51:     puts("You entered 2.");
52: }
53:
54: void other(void)
55: {
56:     puts("You entered something other than 1 or 2.");
57: }
```

该程序的运行情况如下：

```
Enter an integer between 1 and 10, 0 to exit:
2
You entered 2.

Enter an integer between 1 and 10, 0 to exit:
9
You entered something other than 1 or 2.

Enter an integer between 1 and 10, 0 to exit:
0
```

分析：该程序使用了一个死循环（从第 16 行开始），该循环不断执行，直到用户输入了 0。当用户输入一个非零值时，这个值将被传递给 func1()。在 func1()中，第 32 行声明了一个函数指针 ptr。由于是在函数中声明的，因此 ptr 只在 func1()中可见。这是合适的，因为程序的其他部分无需访问它。然后，func1()根据用户输入的值将 ptr 指向合适的函数（第 34～39 行）。第 41 行调用 ptr()，这相当于调用相应的函数。

当然，该程序只是为演示而编写的。不使用函数指针，也能很轻松地完成这样的任务。

下面介绍另一种使用指针来调用不同函数的方式：将指针作为参数传递给函数。程序清单 15.10 是程序清单 15.9 的修订版。

程序清单 15.10　　passptr.c：将指针作为参数传递给函数

```
1:  /* Passing a pointer to a function as an argument. */
2:
3:  #include <stdio.h>
4:
5:  /* The function prototypes. The function func1() takes as */
6:  /* its one argument a pointer to a function that takes no */
7:  /* arguments and has no return value. */
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
8:
9:  void func1(void (*p)(void));
10: void one(void);
11: void two(void);
12: void other(void);
13:
14: int main( void )
15: {
16:     /* The pointer to a function. */
18:     void (*ptr)(void);
19:     int nbr;
20:
21:     for (;;)
22:     {
23:         puts("\nEnter an integer between 1 and 10, 0 to exit: ");
24:         scanf("%d", &nbr);
25:
26:         if (nbr == 0)
27:             break;
28:         else if (nbr == 1)
29:             ptr = one;
30:         else if (nbr == 2)
31:             ptr = two;
32:         else
33:             ptr = other;
34:         func1(ptr);
35:     }
36:     return 0;
37: }
38:
39: void func1(void (*p)(void))
40: {
41:     p();
42: }
43:
44: void one(void)
45: {
46:     puts("You entered 1.");
47: }
48:
49: void two(void)
50: {
51:     puts("You entered 2.");
52: }
53:
54: void other(void)
55: {
56:     puts("You entered something other than 1 or 2.");
57: }
```

该程序的运行情况如下：

```
Enter an integer between 1 and 10, 0 to exit:
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
2
You entered 2.

Enter an integer between 1 and 10, 0 to exit:
11
You entered something other than 1 or 2.

Enter an integer between 1 and 10, 0 to exit:
0
```

分析：请注意程序清单 15.10 和 15.9 之间的差别。函数指针的声明被移到了 main()函数中（第 18 行），因为 main()函数需要使用它。现在，main()函数中的代码根据用户输入的值，将指针初始化为指向相应的函数（第 26～33 行），然后将初始化后的指针传递给 func1()。在程序清单 15.10 中，func1()的唯一作用是调用 ptr 指向的函数。同样，该程序也是为演示而编写的。同样的原理可用于实际程序中（如下一节范例所示）。

一种需要使用函数指针的情况是，必须进行排序。有时候，您可能需要根据不同的条件进行排序。例如，有时候可能要按字母顺序排序，而有时候又要按字母顺序反序排列。通过使用函数指针，程序能够调用相应的函数。更准确地说，通常调用不同的比较函数。

请看程序清单 15.7。在 sort()函数中，实际的排列次序是由库函数 strcmp()的返回值决定的，该函数告诉调用程序，给定的字符串是“小于”还是“大于”另一个字符串。编写两个比较函数——一个按字母顺序排列（即 A 小于 Z），另一个按字母反序排列（即 A 大于 Z）——如何呢？程序可以询问用户要按哪种方式排序，然后排序函数可以使用函数指针来调用相应的比较函数。程序清单 15.11 在程序清单 15.7 的基础上做了修改，添加了这种特性。

程序清单 15.11　　ptrsort.c：使用函数指针来控制排序方式

```
1:  /* Inputs a list of strings from the keyboard, sorts them */
2:  /* in ascending or descending order, and then displays them */
3:  /* on the screen. */
4:  #include <stdlib.h>
5:  #include <stdio.h>
6:  #include <string.h>
7:
8:  #define MAXLINES 25
9:
10: int get_lines(char *lines[]);
11: void sort(char *p[], int n, int sort_type);
12: void print_strings(char *p[], int n);
13: int alpha(char *p1, char *p2);
14: int reverse(char *p1, char *p2);
15:
16: char *lines[MAXLINES];
17:
18: int main( void )
19: {
20:    int number_of_lines, sort_type;
21:
22:    /* Read in the lines from the keyboard. */
23:
24:    number_of_lines = get_lines(lines);
25:
26:    if ( number_of_lines < 0 )
27:    {
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
28:       puts("Memory allocation error");
29:       exit(-1);
30:    }
31:
32:    puts("Enter 0 for reverse order sort, 1 for alphabetical:" );
33:    scanf("%d", &sort_type);
34:
35:    sort(lines, number_of_lines, sort_type);
36:    print_strings(lines, number_of_lines);
37:    return 0;
38: }
39:
40: int get_lines(char *lines[])
41: {
42:     int n = 0;
43:     char buffer[80];  /* Temporary storage for each line. */
44:
45:     puts("Enter one line at time; enter a blank when done.");
46:
47:     while (n < MAXLINES && gets(buffer) != 0 && buffer[0] != '\0')
48:     {
49:        if ((lines[n] = (char *)malloc(strlen(buffer)+1)) == NULL)
50:        return -1;
51:        strcpy( lines[n++], buffer );
52:     }
53:     return n;
54:
55: } /* End of get_lines() */
56:
57: void sort(char *p[], int n, int sort_type)
58: {
59:     int a, b;
60:     char *x;
61:
62:     /* The pointer to function.  */
63:
64:     int (*compare)(char *s1, char *s2);
65:
66:     /* Initialize the pointer to point to the proper comparison */
67:     /* function depending on the argument sort_type. */
68:
69:     compare = (sort_type) ? reverse : alpha;
70:
71:     for (a = 1; a < n; a++)
72:     {
73:        for (b = 0; b < n-1; b++)
74:        {
75:           if (compare(p[b], p[b+1]) > 0)
76:           {
77:              x = p[b];
78:              p[b] = p[b+1];
79:              p[b+1] = x;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
80:             }
81:         }
82:     }
83: }  /* end of sort() */
84:
85: void print_strings(char *p[], int n)
86: {
87:     int count;
88:
89:     for (count = 0; count < n; count++)
90:         printf("%s\n", p[count]);
91: }
92:
93: int alpha(char *p1, char *p2)
94: /* Alphabetical comparison. */
95: {
96:     return(strcmp(p2, p1));
97: }
98:
99: int reverse(char *p1, char *p2)
100: /* Reverse alphabetical comparison. */
101: {
102:     return(strcmp(p1, p2));
103: }
```

该程序的运行情况如下：

```
Enter one line at time; enter a blank when done.
Roses are red
Violets are blue
C has been around,
But it is new to you!

Enter 0 for reverse order sort, 1 for alphabetical:
0

Violets are blue
Roses are red
C has been around,
But it is new to you!
```

分析：main()函数中的第 32 和 33 行询问用户要按哪种方式排序。用户输入的值被存储在 sort_type 中。然后，将这个值以及其他信息传递给函数 sort()。对 sort()函数做了两处修改。第 64 行声明了一个名为 compare 的函数指针，它接受两个字符指针（字符串）作为参数。第 69 行根据 sort_type 的值，将两个新增的函数之一赋给 compare。这两个新增的函数是 alpha()和 reverse()。前者像程序清单 15.7 那样使用库函数 strcmp()；而后者则将传递来的两个参数的位置互换，从而实现反序排列。

应　读	不 应 读
声明函数指针时，一定要使用圆括号。 下面的代码声明一个指向某个函数的指针，该函数不接受任何参数，且返回一个字符： `char (*func)( );` 下面的代码声明一个返回 char 指针的函数： `char *func( );`	初始化指针之前，不要使用它； 不要将函数指针指向返回类型和参数列表与之不匹配的函数。

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

15.5　链　表

链表（linked list）是一种很有用的数据存储方式，在C语言中很容易实现。为何在介绍指针时讨论链表呢？因为指针是实现链表的核心所在。

链表有多种，包括单向链表、双向链表和二叉树。每种链表适用于不同的数据存储类型。它们之间的共同点之一是，数据项之间的链接是使用数据项中的信息以指针的方式来定义的。这与数组完全不同，在数组中，数据项之间的链接是数组的存储和排列方式形成的。本节介绍最基本的链表：单向链表（我们称之为简单链表）。

15.5.1　有关链表的基本知识

链表中的数据项被存储在一个结构中（第 11 天的课程介绍了结构）。该结构包含用于存储数据的数据元素——具体是什么元素取决于程序的需求。另外，还包含一个指针元素，该指针用于提供链表中的链接。下面是一个简单的例子：

```
struct person {
char name[20];
struct person *next;
};
```

上述代码定义了一种名为 person 的结构。为存储数据，person 结构只包含一个长度为 20 的字符数组。对于这样简单的数据，通常不会使用链表，这里只是作为一个例子。person 结构还包含一个 person 指针，即一个指向另一个 person 结构的指针。这意味着每个 person 结构不但可以存储一系列的数据，还可以存储一个指向另一个 person 结构的指针。图 15.7 说明了这是如何将链表连接到一起的。

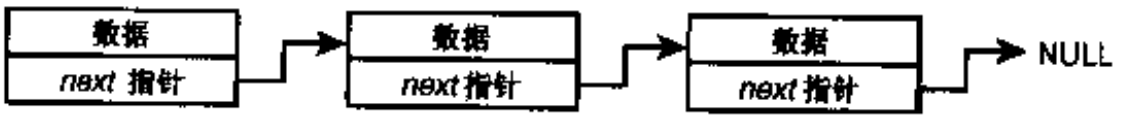


图 15.7　链表中的链接

在图 15.7 中，每个 person 结构都指向下一个 person 结构。最后一个 person 结构没有指向任何东西。链表的最后一个节点是通过将其指针元素设置为 NULL 来标识的。



注意：组成链表的结构可被称为链接、节点或元素。

您已经知道链表的最后一个节点是如何标识的。第一个节点又是如何标识的呢？这是通过一个名为头指针（head pointer）的特殊指针（不是结构）标识的。头指针总是指向链表的第一个节点；而第一个节点包含一个指向第二个节点的指针，第二个节点包含一个指向第三个节点的指针，依此类推，最后一个节点包含的指针的值为 NULL。如果整个链表为空（即不包含任何节点），则将头指针设置为 NULL。图 15.8 说明了添加第一个节点前后，头指针的情况。

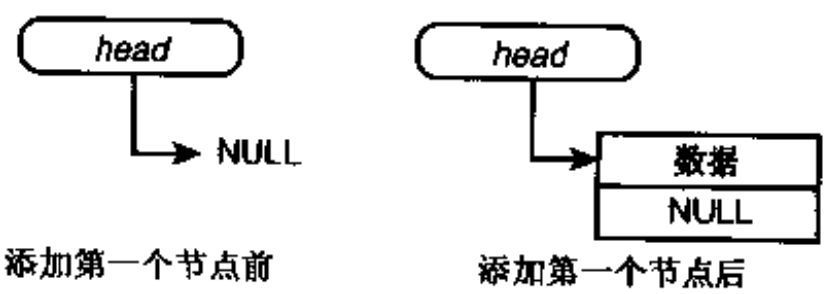


图 15.8　链表的头指针

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com



注意：头指针是指向链表的第一个节点的指针。头指针有时也被称为首节点指针或顶指针（top pointer）。

15.5.2 使用链表

使用链表时，可以添加、删除、修改节点。修改节点并不难，但添加和删除节点有些麻烦。正如前面指出的，链表中的节点是通过指针连接起来的。添加和删除节点的大部分工作是处理指针。节点可能被添加到链表的开始、中间或末尾，这决定了应如何修改指针。

今天课程的后面将演示一个简单的链表，并编写一个复杂的程序。在介绍代码的细节之前，先来看一些您需要对链表执行的操作。在接下来的几节中，将继续使用前面介绍的 person 结构。

1. 准备工作

使用链表之前，必须定义一个结构（用于构成链表），还需要声明一个头指针。由于开始时链表为空，因此应将头指针初始化为 NULL。还需要一个类型为链表结构的指针，用于添加节点（可能需要多个这样的指针）。代码如下：

```
struct person {
    char name[20];
    struct person *next;
};
struct person *new;
struct person *head;
head = NULL;
```

2. 将节点加入到链表的开头

如果头指针为 NULL，说明链表原先为空，新增的节点是链表中唯一的一个节点。如果头指针不为 NULL，则说明链表已经包含一个或多个节点。然而，无论是哪种情况，将一个新的节点加入到链表开头的步骤都相同：

1. 创建一个结构实例：使用 malloc()来分配内存空间；

2. 将新节点的 next 指针设置为头指针的值——如果链表原先为空，则为 NULL；否则为第一个节点的地址；

3. 让头指针指向新的节点。

完成上述任务的代码如下：

```
new = (person*)malloc(sizeof(struct person));
new->next = head;
head = new
```

注意，对 malloc()的返回值进行了强制类型转换，使之为正确的类型——一个指向 person 结构的指针。



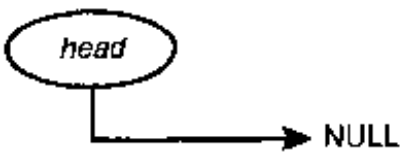
警告：以正确的次序切换指针至关重要。如果先给头指针重新赋值，则链表将丢失。

图 15.9 说明了在空链表中新添节点的步骤；而图 15.10 说明了在将新节点加入到非空链表的开头的步骤。

鸿鹄论坛 http://bbs.hh010.com

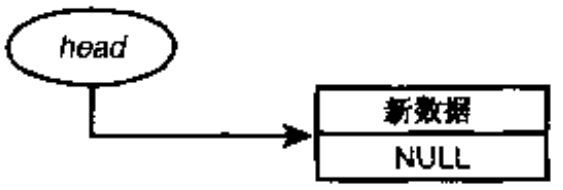
www.plcworld.cn

鸿鹄书院 http://book.hh010.com



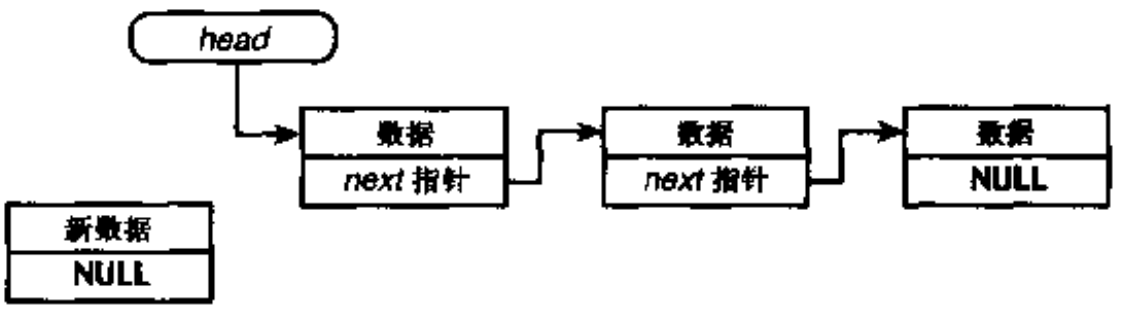


添加前

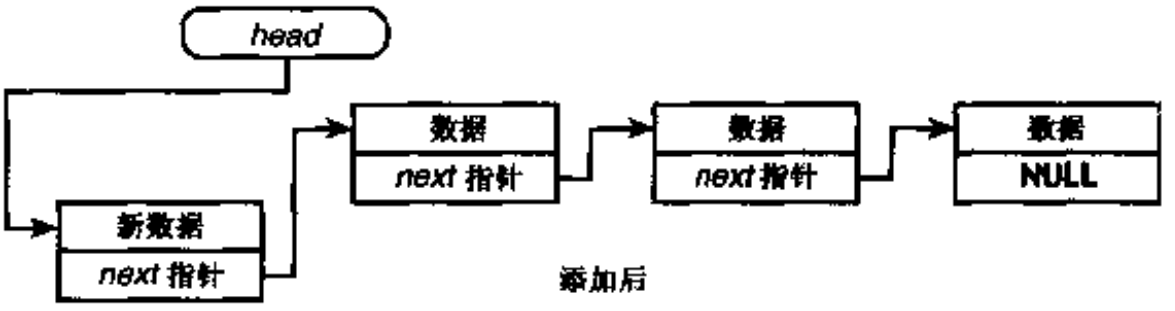


添加后

图 15.9　在空链表中新增一个节点



添加前



添加后

图 15.10　将一个新节点加入到非空链表的开头

注意，这里使用 malloc()来为新节点分配内存。每次添加新节点时，只分配该节点所需的内存。也可以使用 calloc()函数。有必要指出这两个函数之间的区别，主要区别在于，calloc()会初始化新节点，而 malloc()不会。



警告：在上述代码片段中，并没有检查 malloc()的返回值，以确保成功地分配了内存。在实际的程序中，一定要检查内存分配函数的返回值。



提示：在声明指针时，应将其初始化为 NULL。不要不初始化指针，这样做无疑是自找麻烦。

3. 将节点加入到链表末尾

要将节点加入到链表末尾，应从头指针开始往下找，直到到达最后一个节点。然后执行下面的步骤：

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

1. 创建一个结构实例：使用 malloc()来分配内存空间；
2. 让最后一个节点的 next 指针指向新节点（malloc()返回的地址）；
3. 将新节点的 next 指针设置为 NULL，以表明它是链表中的最后一个节点。

代码如下：

```
person *current;
...
current = head;
while (current->next != NULL)
        current = current->next;
new = (person*)malloc(sizeof(struct person));
current->next = new;
new->next = NULL;
```

图 15.11 说明了将节点添加到链表末尾的步骤。

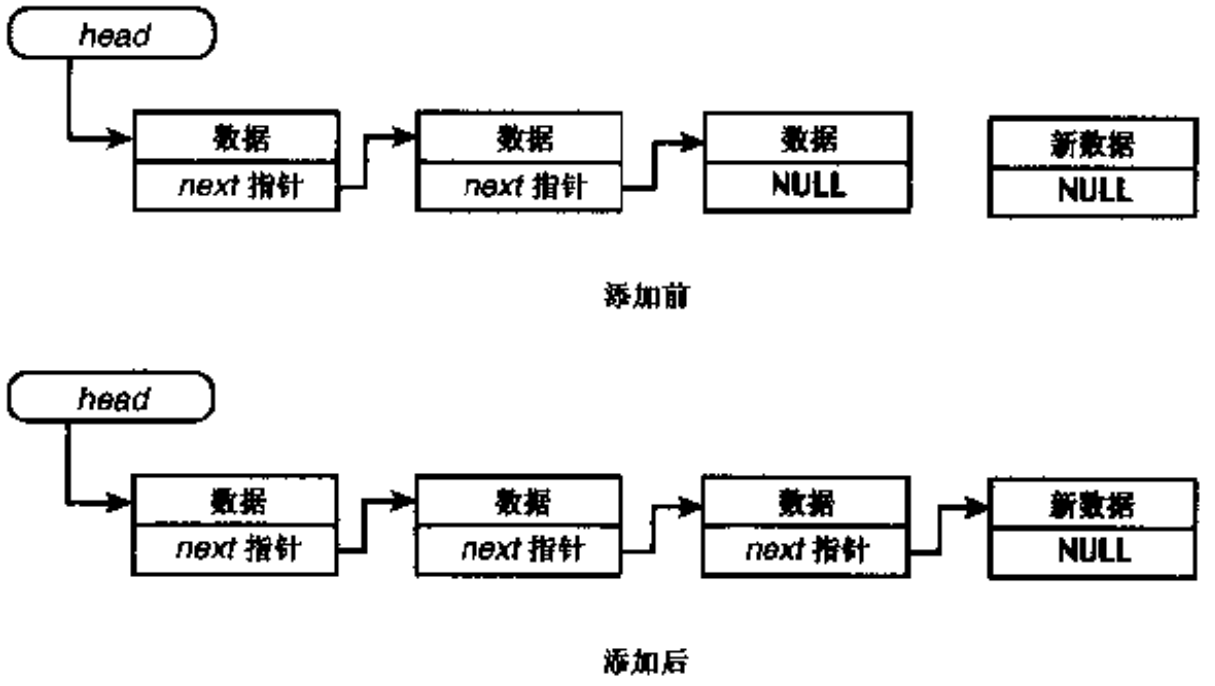


图 15.11　将节点添加到链表的末尾

4. 将节点添加到链表的中间

使用链表时，大多数时候需要将节点加入到链表的中间。新节点将被加入到的具体位置取决于您是如何维护链表的——例如，是否按一个或多个数据元素进行排序。因此，首先要确定新节点应被加入到链表的什么位置，然后再加入。步骤如下：

1. 在链表中，确定新元素应被加入到哪个节点的后面。我们将其称为标记节点；
2. 创建一个结构实例：使用 malloc()来分配内存空间；
3. 将标记节点的 next 指针指向新元素（其地址是由 malloc()返回的）；
4. 将新节点的 next 指针指向标记节点指向的节点。

代码如下：

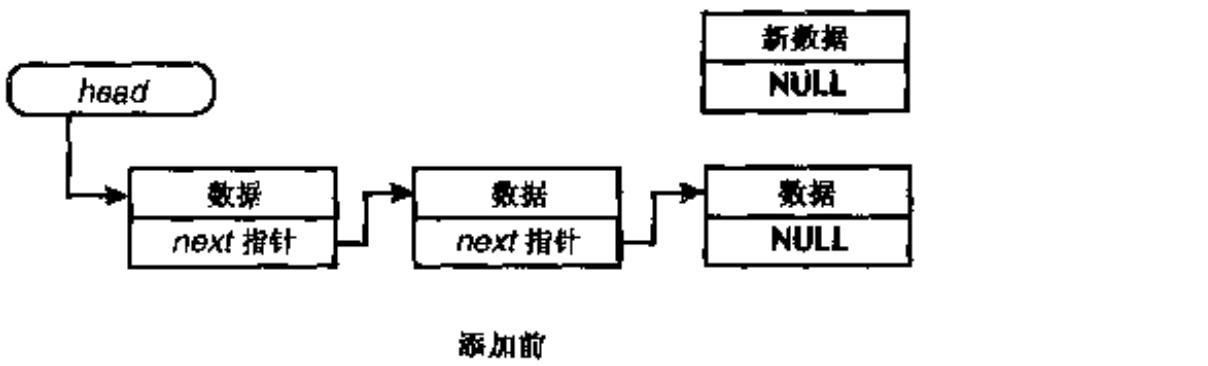
```
person *marker;
/* Code here to set marker to point to the desired list location. */
...
new = (LINK)malloc(sizeof(PERSON));
new->next = marker->next;
marker->next = new;
```

图 15.12 说明了这一过程。

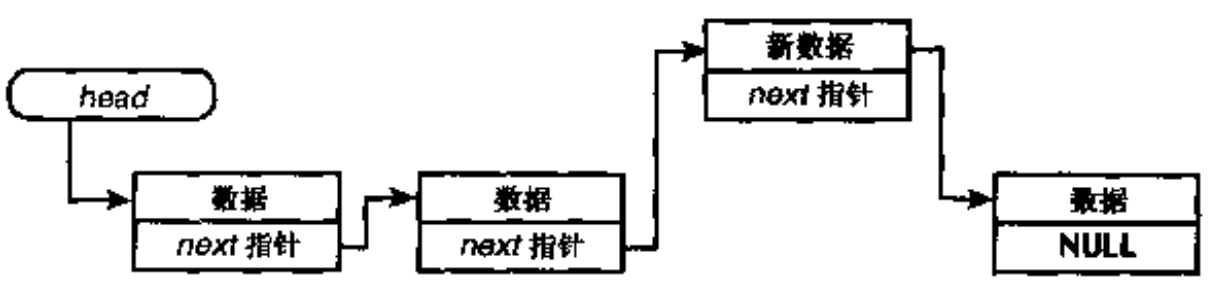
鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com



添加前



添加后

图 15.12　将节点加入到链表的中间

5. 删除节点

删除链表中的节点时，对指针的操纵很简单。具体的步骤取决于节点位于列表的什么位置。

- 要删除第一个节点，将头指针指向列表的第二个节点；
- 要删除最后一个节点，将倒数第二个节点的 next 指针设置为 NULL；
- 要删除其他节点，将要删除的节点的前一个节点的 next 指针指向要删除的节点的后一个节点。

另外，对于从链表中被删除的节点，应释放它占用的内存，以防程序占用不需要的内存（被称为内存泄漏）。释放内存是使用 free()函数来完成的，有关该函数的详细信息将在第 20 天的课程中介绍。删除链表中第一个节点的代码如下：

```
free(head);
head = head->next;
```

删除链表中最后一个节点的代码如下：

```
person *current1, *current2;
current1 = head;
current2= current1->next;
while (current2->next != NULL)
{
    current1 = current2;
    current2= current1->next;
}
free(current1->next);
current1->next = null;
if (head == current1)
        head = null;
```

最后，删除链表中间的节点的代码如下：

```
person *current1, *current2;
/* Code goes here to have current1 point to the */
/* element just before the one to be deleted. */
current2 = current1->next;
free(current1->next);
current1->next = current2->next;
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com

完成上述步骤后，删除的节点仍留在内存中，但不再在链表中，因为链表中没有任何指针指向它。在实际的程序中，应归还被删除的节点占用的内存，这是使用 free()函数来完成的，有关该函数的细节，将在第 20 天的课程中介绍。

15.5.3 演示简单链表

程序清单 15.12 演示了链表的基本用法。该程序是为了演示而编写的，因为它不读取用户输入，同时除了说明完成最基本的链表任务所需的代码外，并没有完成任何有用的工作。该程序完成以下工作：

1. 定义使用链表所需的结构和指针；
2. 将一个节点添加到链表开头；
3. 将一个节点添加到链表末尾；
4. 将一个节点添加到链表中间；
5. 将链表的内容显示到屏幕上。

程序清单 15.12　　linkdemo.c：链表的基本用法

```
1:  /* Demonstrates the fundamentals of using */
2:  /* a linked list. */
3:
4:  #include <stdlib.h>
5:  #include <stdio.h>
6:  #include <string.h>
7:
8:  /* The list data structure. */
9:  struct data {
10:     char name[20];
11:     struct data *next;
12: };
13:
14: /* Define typedefs for the structure */
15: /* and a pointer to it. */
16: typedef struct data PERSON;
17: typedef PERSON *LINK;
18:
19: int main( void )
20: {
21:     /* Head, new, and current element pointers. */
22:     LINK head = NULL;
23:     LINK new = NULL;
24:     LINK current = NULL;
25:
26:     /* Add the first list element. We do not */
27:     /* assume the list is empty, although in */
28:     /* this demo program it always will be. */
29:
30:     new = (LINK)malloc(sizeof(PERSON));
31:     new->next = head;
32:     head = new;
33:     strcpy(new->name, "Abigail");
34:
35:     /* Add an element to the end of the list. */
36:     /* We assume the list contains at least one element. */
```


鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

鸿鹄书院 http://book.hh010.com


```
37:
38:   current = head;
39:   while (current >next != NULL)
40:   {
41:     current = current->next;
42:   }
43:
44:   new = (LINK)malloc(sizeof(PERSON));
45:   current->next = new;
46:   new->next = NULL;
47:   strcpy(new->name, "Carolyn");
48:
49:   /* Add a new element at the second position in the list. */
50:   new = (LINK)malloc(sizeof(PERSON));
51:   new->next = head->next;
52:   head->next = new;
53:   strcpy(new->name, "Beatrice");
54:
55:   /* Print all data items in order. */
56:   current = head;
57:   while (current != NULL)
58:   {
59:     printf("\n%s", current->name);
60:     current = current->next;
61:   }
62:
63:   printf("\n");
64:
65:   return 0;
66: }
```

该程序的输出如下：

```
Abigail
Beatrice
Carolyn
```

分析：您至少能够理解其中的一些代码。第 9～12 行声明了用于组成链表的结构。第 16 和 17 行为该结构以及指向这种结构的指针定义了 typedef。严格地说，这是不必要的，但这样能够让您用 PERSON 和 LINK，而不必使用 struct data 和 struct data *，从而简化了代码编写工作。

第 22～24 行声明了 head 指针以及操纵链表时需要使用的其他指针。这些指针都被初始化为 NULL。

第 30～33 行将一个新节点加入到链表的开头。第 30 行创建一个新的结构，并为它分配内存。这里假设 malloc()能够成功地分配内存——在实际的程序中决不要这样做。

第 31 行将新创建的结构的 next 指针设置为 head 指针的值。为何不简单地将其设置为 NULL 呢？仅当您知道链表为空时，这样做才有效；而像程序清单中那样编写代码，即使链表中已经包含一些元素，也管用。这样，新增的节点将指向原来的第一个节点，这正是您希望的。

第 32 行让 head 指针指向新节点，而第 33 行将一些数据存储到该节点中。

将节点加入到链表末尾要复杂些。虽然在这个例子中，链表中只包含一个节点，但在实际的程序中，不能做这样的假设。因此，必须从第一个节点开始遍历链表，直到找到最后一个节点（其 next 指针为 NULL）。这项任务是在第 38～42 行完成的。找到最后一个节点后，事情就简单了——只需创建一个新的结构实例，让最后一个节点指向它，并将新节点的 next 指针设置为 NULL（因为这是链表的最后一个节点）。这项工作是

鸿鹄论坛 http://bbs.hh010.com

www.plcworld.cn

在第 44～47 行完成的。

注意，这里将 malloc()的返回值强制转换为 LINK 类型（有关强制类型转换的更详细的信息，请参阅第 20 天的课程）。

接下来要将一个节点加入到列表中间——在这个例子中，是第二个位置。创建新的结构实例（第 50 行）后，将新节点的 next 指针指向原来的第二个节点（现在为第三个节点，第 51 行），并将第一个节点的 next 指针指向新的节点（第 52 行）。

最后，程序打印链表中的所有节点。这很简单，只需从 head 指针指向的节点开始，然后遍历整个链表，直到到达最后一个节点（其 next 指针为 NULL）。这项任务是在第 56～61 行完成的。

15.5.4 实现链表

知道如何将节点加入到链表中后，下面介绍如何使用链表。程序清单 15.13 中的程序很长，它使用链表来存储 5 个字符。当然可以不存储字符，而存储姓名、地址或其他数据。为使该范例尽可能简单，每个节点只存储一个字符。

该程序之所以复杂，是由于它在加入节点的同时，对节点进行排序。当然，这也使该程序很有用。节点被加入到链表的开头、中间还是末尾取决于它的值。该链表总是按顺序排列的。如果编写一个总是将节点加入到链表末尾的程序，逻辑则简单得多，但用处也将小得多。

程序清单 15.13　　linklist.c：实现一个字符链表

```
1:  /*=================================================================    *
2:   * Program:  listlist.c                                               *
3:   * Book:     Sams Teach Yourself C in 21 Days                         *
4:   * Purpose:  Implementing a linked list                               *
5:   *=================================================================    */
6:  #include <stdio.h>
7:  #include <stdlib.h>
8:
9:  #ifndef NULL
10:   #define NULL 0
11: #endif
12:
13: /* List data structure */
14: struct list
15: {
16:   int    ch;     /* using an int to hold a char */
17:   struct list *next_rec;
18: };
19:
20: /* Typedefs for the structure and pointer. */
21: typedef struct list LIST;
22: typedef LIST *LISTPTR;
23:
24: /* Function prototypes. */
25: LISTPTR add_to_list( int, LISTPTR );
26: void show_list(LISTPTR);
27: void free_memory_list(LISTPTR);
28:
29: int main( void )
30: {
31:   LISTPTR first = NULL;  /* head pointer */
```

www.plcworld.cn


```
32:    int i = 0;
33:    int ch;
34:    char trash[256];       /* to clear stdin buffer. */
35:
36:    while ( i++ < 5 )      /* build a list based on 5 items given */
37:    {
38:       ch = 0;
39:       printf("\nEnter character %d, ", i);
40:
41:       do
42:       {
43:          printf("\nMust be a to z: ");
44:          ch = getc(stdin);  /* get next char in buffer  */
45:          gets(trash);       /* remove trash from buffer */
46:       } while( ch < 'a' || ch >'z')&&(ch <'A'//ch > 'z');
47:
48:       first = add_to_list( ch, first );
49:    }
50:
51:    show_list( first );          /* Dumps the entire list */
52:    free_memory_list( first );    /* Release all memory */
53:    return 0;
54: }
55:
56: /*=========================================================*
57:  * Function: add_to_list()
58:  * Purpose : Inserts new link in the list
59:  * Entry   : int ch = character to store
60:  *           LISTPTR first = address of original head pointer
61:  * Returns : Address of head pointer (first)
62:  *=========================================================*/
63:
64: LISTPTR add_to_list( int ch, LISTPTR first )
65: {
66:    LISTPTR new_rec = NULL;       /* Holds address of new rec */
67:    LISTPTR tmp_rec = NULL;       /* Hold tmp pointer         */
68:    LISTPTR prev_rec = NULL;
69:
70:    /* Allocate memory. */
71:    new_rec = (LISTPTR)malloc(sizeof(LIST));
72:    if (!new_rec)      /* Unable to allocate memory */
73:    {
74:       printf("\nUnable to allocate memory!\n");
75:       exit(1);
76:    }
77:
78:    /* set new link's data */
79:    new_rec->ch = ch;
80:    new_rec->next_rec = NULL;
81:
82:    if (first == NULL)   /* adding first link to list */
83:    {
```

www.plcworld.cn


```
84:       first = new_rec;
85:       new_rec->next_rec = NULL;  /* redundant but safe */
86:    }
87:    else   /* not first record */
88:    {
89:       /* see if it goes before the first link */
90:       if ( new_rec->ch < first->ch)
91:       {
92:          new_rec->next_rec = first;
93:          first = new_rec;
94:       }
95:       else   /* it is being added to the middle or end */
96:       {
97:          tmp_rec = first->next_rec;
98:          prev_rec = first;
99:
100:         /* Check to see where link is added. */
101:
102:         if ( tmp_rec == NULL )
103:         {
104:            /* we are adding second record to end */
105:            prev_rec->next_rec = new_rec;
106:         }
107:         else
108:         {
109:            /* check to see if adding in middle */
110:            while (( tmp_rec->next_rec != NULL))
111:            {
112:               if( new_rec->ch < tmp_rec->ch )
113:               {
114:                  new_rec->next_rec = tmp_rec;
115:                  if (new_rec->next_rec != prev_rec->next_rec)
116:                  {
117:                     printf("ERROR");
118:                     getc(stdin);
119:                     exit(0);
120:                  }
121:                  prev_rec->next_rec = new_rec;
122:                  break;   /* link is added: exit while */
123:               }
124:               else
125:               {
126:                  tmp_rec = tmp_rec->next_rec;
127:                  prev_rec = prev_rec->next_rec;
128:               }
129:            }
130:
131:            /* check to see if adding to the end */
132:            if (tmp_rec->next_rec == NULL)
133:            {
134:               if (new_rec->ch < tmp_rec->ch ) /* 1 b4 end */
135:               {
```

www.plcworld.cn


```
136:                new_rec->next_rec = tmp_rec;
137:                prev_rec->next_rec = new_rec;
138:             }
139:             else  /* at the end */
140:             {
141:                tmp_rec->next_rec = new_rec;
142:                new_rec->next_rec = NULL;  /* redundant */
143:             }
144:          }
145:       }
146:    }
147:    return(first);
148: }
149:
150:
151: /*=================================================*
152:  * Function: show_list
153:  * Purpose : Displays the information current in the list
154:  *=================================================*/
155:
156: void show_list( LISTPTR first )
157: {
158:    LISTPTR cur_ptr;
159:    int counter = 1;
160:
161:    printf("\n\nRec addr  Position  Data  Next Rec addr\n");
162:    printf("========  ========  ====  ============\n");
163:
164:    cur_ptr = first;
165:    while (cur_ptr != NULL )
166:    {
167:       printf(" %X   ", cur_ptr );
168:       printf("    %2i      %c", counter++, cur_ptr->ch);
169:       printf("     %X   \n",cur_ptr->next_rec);
170:       cur_ptr = cur_ptr->next_rec;
171:    }
172: }
173:
174: /*=================================================*
175:  * Function: free_memory_list
176:  * Purpose : Frees up all the memory collected for list
177:  *=================================================*/
178:
179: void free_memory_list(LISTPTR first)
180: {
181:    LISTPTR cur_ptr, next_rec;
182:    cur_ptr = first;                /* Start at beginning */
183:
184:    while (cur_ptr != NULL)         /* Go while not end of list */
185:    {
186:       next_rec = cur_ptr->next_rec; /* Get address of next record */
187:       free(cur_ptr);               /* Free current record */
```

www.plcworld.cn


```
188:      cur_ptr -> next_rec;          /* Adjust current record*/
189:   }
190: }
```

该程序的运行情况如下：

```
Enter character 1,
Must be a to z: q

Enter character 2,
Must be a to z: b

Enter character 3,
Must be a to z: z

Enter character 4,
Must be a to z: c

Enter character 5,
Must be a to z: a

Rec addr  Position  Data  Next Rec addr
--------  --------  ----  -------------
    2224A0        1     a     222470
    222470        2     b     222490
    222490        3     c     222450
    222450        4     q     222480
    222480        5     z     0
```



注意：您运行该程序时，显示的地址可能不同。

分析：该程序演示了如何将节点加入到链表中。该程序清单理解起来不那么容易，但如果详细查看，你会发现它使用了前面讨论的三种加入节点的方法。该程序可将节点加入到链表的开头、中间或末尾。另外，它还考虑到了加入第一个节点（被加入到开头）和第二个节点（被加入到中间）的特殊情况。



提示：要完全理解该程序清单，最简单的方式是在编译器的调试器中逐行执行它，并阅读下面的分析。通过了解执行的逻辑，您将更深入地理解该程序清单。

在程序清单 15.13 中，开始的一些内容您应该熟悉或易于理解。第 9～11 行检查是否定义了 NULL 的值。如果没有，则将其定义为 0（第 10 行）。第 14～22 行定义用于构成链表的结构，同时声明了类型定义，以便使用这种结构和指针时更容易。

main()函数很容易理解。第 31 行声明了头指针 first，并将其初始化为 NULL。前面说过，决不要不初始化指针。第 36～49 行包含一个 while 循环，用于从用户那里读取 5 个字符。在这个外部循环（它执行 5 次）中，使用了一个 do...while 循环来确保用户输入的是字母。这里也可以直接使用 isalpha()函数，这样更简单。

获得一个字符后，调用 add_to_list()，并将指向链表开头的指针和要加入的数据传递给它。

最后，main()函数调用 show_list()和 free_memory-list()，前者显示链表中的数据，后者释放用于存储链表中的节点的内存。这两个函数的工作原理类似，它们都从链表的开始位置开始（通过头指针 first），并使用

www.plcworld.cn

一个 while 循环来遍历整个链表中的节点（通过 next_ptr）。当 next_ptr 为 NULL 时，说明到达了链表末尾，因此函数结束。

在该程序清单中，最重要（也是最复杂）的函数是第 56～149 行的 add_to_list()函数。第 66～68 行声明了三个指针，用于指向三个不同的节点。new_rec 指针指向要加入的新节点；tem_pointer 指向链表中的当前节点。如果链表中包含多个节点，则 prev_rec 指针用于指向当前节点的前一个节点。

第 71 行为新增的节点分配内存。new_rec 指针被设置为 malloc()返回的值。如果无法分配内存，则打印错误消息，然后退出程序（第 74 和 75 行）；如果成功地分配了内存，则程序继续执行。

第 79 行将结构中的数据设置为传递给函数的数据。这里只是将传递给函数的值 ch 赋给新节点的字符元素（new_rec->ch），而更复杂的程序中，可能需要给多个元素赋值。第 80 行将新节点的 next_rec 设置为 NULL，避免它随机地指向某个位置。

第 82 行开始加入节点，它检查链表是否为空。如果链表为空，即头指针 first 为 NULL，则只需将头指针指向新节点即可。

如果链表不为空，则执行 else 语句（第 87 行）。第 90 行检查新节点是否应该加入到链表的开头。您可能还记得，这是三种加入节点的情形之一。如果新节点应被加入到开头，则将新节点的 next_rec 指针指向原来的第一个节点（第 92 行），然后将头指针 first 指向新节点（第 93 行）。这样新节点便被加入到链表的开头。

如果链表不为空，且新节点不应加入到链表开头，则它将被加入到链表中间或末尾。因此，第 97 和 98 行分别将前面声明的 tmp_rec 和 prev_rec 指针指向第二个节点和第一个节点。

如果链表中只有一个节点，则 tmp_rec 将为 NULL。这是因为 tmp_rec 被设置为第一个节点的 next_ptr，而后者等于 NULL。第 102 行检查这种特殊情况。如果 tmp_rec 为 NULL，则说明新节点是链表的第二个节点。由于您已经知道新节点不应加入到第一个节点的前面，因此只可能加入到链表的末尾。要完成这样的任务，只需将 prev_rec->next_ptr 指向新节点即可。

如果 tmp_rec 不为 NULL，则说明链表中已经有多个节点。因此第 110～129 行的 while 语句遍历余下的节点，以确定新节点应加入到什么位置。第 112 行检查新节点的数据值是否小于当前节点。如果是，则说明应将新节点加入到当前节点的前面；否则进入下一个节点。第 126 和 127 行分别将指针 tmp_rec 和 prev_rec 指向相应的下一个节点。

如果新输入的字符"小于"当前节点中的字符，则按本章前面介绍的逻辑将新节点加入到链表中间。这是在第 114～122 行完成的。第 114 行将新节点的 next_rec 指针指向当前节点（tmp_rec）。第 121 行将前一个节点的 next_rec 指针指向新节点。这样便完成了。这里使用一条 break 语句来跳出 while 循环。



注意：第 115～120 行是调试代码，可以删除它们（只要程序正确地运行，这些代码便不会被执行）。将新节点的 next_rec 指针指向当前节点后，该指针应该与前一个节点的 next_rec（也指向当前节点）相同。如果不同，则说明出了问题。

上面介绍的是将节点加入到链表中间的逻辑。如果到达了链表的末尾，则第 110～129 行的 while 语句会结束运行，而不会将节点加入。将节点加入到链表末尾的工作是由第 132～144 行的代码完成的。

到达链表的最后一个节点后，tmp_rec->next_rec 将为 NULL，第 132 行检测这种条件。第 134 行判断新节点应被加入到最后一个节点前还是后。如果是最后一个节点后，则将最后一个节点的 next_rec 指向新节点（第 132 行），并将新节点的 next_rec 指针设置为 NULL（第 142 行）。

1．改进程序清单 15.13

链表不太好懂。然而从程序清单 15.13 可知，链表是一种按顺序存储数据的好方式。由于将新的数据项添加到链表中很容易，因此使用链表按顺序存储一系列数据项，比使用诸如数组等数据结构更简单。可以对该程序清单进行修改，用于对姓名、电话号码或其他任何数据进行排序。另外，虽然该程序清单对数据按升序排列（A 到 Z），但按降序排列（Z 到 A）也一样容易。

www.plcworld.cn

2. 删除链表中的节点

将信息添加到链表中的功能是必不可少的，但有时候您可能还想删除链表中的信息。删除节点与添加节点类似，也可以删除链表开头、中间和末尾的节点。无论是哪种情况，都必须对相应的指针进行调整。另外，还必须释放被删除的节点占用的内存。



注意：删除节点后，别忘了释放它占用的内存。

应　该	不 应 该
一定要理解 calloc()和 malloc()之间的差别。最重要的是一定要记住，malloc()不对分配的内存进行初始化，而 calloc()会。	删除节点时，别忘了释放它占用的内存。

15.6 总　结

今天的课程介绍了指针的一些高级用法。您可能已经意识到，指针是 C 语言的核心部分。不使用指针的 C 程序非常少。您知道了如何使用指向指针的指针，以及在处理字符串时，指针数组很有用。您还知道 C 语言将多维数组视为由数组组成的数组，并知道如何将指针用于这种数组。您学习了如何声明和使用函数指针——一种灵活而重要的编程工具。最后，您学习了如何实现链表——一种灵活、强大的数据存储方法。

今天的课程很长，也很复杂。虽然其中介绍的一些主题很复杂，但也很有趣。学完本章后，您便真正涉足到了 C 语言一些复杂的功能。C 语言之所以如此流行，功能强大和灵活是其中的两个主要原因。

15.7 问与答

问：指向指针的指针最多可以有多少层？

答：您可以参阅编译器用户手册，了解对此是否有限制。通常，没有理由超过三层（即指向指针的指针的指针）。大多数程序都不会超过两层。

问：指向字符串的指针和指向字符数组的指针有区别吗？

答：没有。字符串可被视为一个字符数组。

问：为充分利用 C 语言，必须应用今天课程介绍的概念吗？

答：可以不应用任何有关指针的高级概念，但将无法充分利用 C 语言提供的强大功能。诸如今天课程介绍的指针操作可以简化很多编程任务，并提高效率。

问：还有函数指针很有用的其他情形吗？

答：有。函数指针还可用于处理菜单。根据用户选择的菜单项，将函数指针设置为指向相应的函数。

问：指出链表的两种优点。

答：首先链表的长度可以在程序运行期间伸缩，而无需在编写代码时预定义；其次，让链表中的节点按顺序排列很容易，因为将添加和删除任何位置的节点都很容易。

15.8 作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

www.plcworld.cn

15.8.1　小测验

1. 编写这样的代码：声明一个 float 变量，声明一个 float 指针并将其初始化为指向 float 变量，声明一个指针并将其初始化为指向 float 指针的指针。

2. 假设您要使用指向指针的指针来将 100 赋给变量 x，下述赋值语句是否正确？

```
*ppx =100;
```

如果不正确，应如何修改？

3. 假设像下面这样声明了一个数组：

```
int array[2][3][4];
```

在编译器看来，该数组的结构是什么样的？

4. 对于问题 3 中声明的数组，表达式 array[0][0]指的是什么？

5. 还是以问题 3 中的数组为例，下述哪些比较的结果为真？

```
array[0][0] == &array[0][0][0];
array[0][1] == array[0][0][1];
array[0][1] == &array[0][1][0];
```

6. 对于接受一个 char 指针数组为参数，且返回 void 的函数，编写其函数原型。

7. 问题 6 中的函数如何知道传递给它的指针数组包含多少个元素？

8. 函数指针是什么？

9. 声明一个这样的函数指针，即返回类型为 char，且接受一个 char 指针数组作为参数。

10. 对于问题 9，下面的答案有什么错误？

```
char *ptr(char *x[]);
```

11. 定义用于链表的结构时，必须包含的一个元素是什么？

12. 头指针为 NULL 意味着什么？

13. 单向链表是如何连接的？

14. 下列语句声明的分别是什么？

 a. `int *var1;`

 b. `int var2;`

 c. `int **var3;`

15. 下列语句声明的分别是什么？

 a. `int a[3][12];`

 b. `int (*b)[12];`

 c. `int *c[12];`

16. 下列语句声明的分别是什么？

 a. `char *z[10];`

 b. `char *y(int field);`

 c. `char (*x)(int field);`

15.8.2　练习

1. 声明一个这样的函数指针，即接受一个 int 参数，且返回一个 float 值。

2. 声明一个函数指针数组，其元素用于指向这样的函数，即接受一个字符串参数，且返回一个 int 值。这种数组有何用途？

3. 声明一个包含 10 个元素的 char 指针数组。

4. 排错：下列代码有何错误？

```
int x[3][12];
int *ptr[12];
ptr = x;
```

www.plcworld.cn

5. 编写一个供单向链表使用的结构。该结构用于存储您的朋友的姓名和地址。

由于下面的练习有多种可能的解决方案，因此附录 F 没有提供它们的答案。

6. 选做题：编写程序，它声明一个每维长度都为 12 的二维数组，并将 X 存储到每个元素中，然后使用一个指向该数组的指针，以网格方式将数组中的值打印到屏幕上。

7. 选做题：编写一个程序，使用 double 指针从用户那里获得 10 个数字，然后对它们进行排序，并将它们打印到屏幕上（提示：请参阅程序清单 15.10）。

8. 选做题：对练习 7 中编写的程序进行修改，让用户指定按升序还是降序排列。

www.plcworld.cn

第 16 天课程　使用磁盘文件

您编写的大多数程序都将使用硬盘中的文件。这些文件包含某种用途的信息：数据存储、配置信息等。今天的课程介绍以下内容：

- 如果将流与磁盘文件关联起来？
- 两种磁盘文件；
- 用于打开文件的命令；
- 如何将数据写入到文件中？
- 如何读取文件中的数据？
- 何时关闭文件？
- 磁盘文件管理；
- 使用临时文件。

16.1　将流与磁盘文件关联起来

第 14 天的课程介绍过，C 语言通过流来执行所有的输入和输出（包括磁盘文件）。您已经知道如何使用 C 语言中与诸如键盘、屏幕和打印机（在有些系统中）等相连的预定义流。磁盘文件流的工作原理也没什么不同。这是流输入/输出的优点之一：用于一种流的技术可以用于其他流，而只需做少量的修改或根本无需修改。使用磁盘文件流时，主要的不同在于，必须明确地创建一个与特定的磁盘文件相关联的流。

16.2　磁盘文件的类型

第 14 天的课程介绍过，C 语言中的流分两类：文本流和二进制流。可以将其中的任何一种流与文件关联起来，然而要将正确的模式用于文件，必须了解这两种流之间的差别。

文本流与文本-模式的文件相关联。文本、模式文件由字符行序列组成，其中每行包含 0 个或更多的字符，并以一个或多个标记行尾的字符结尾。请记住，文本-模式文件中的“行”不同于字符串，它不以空字符（\0）结尾。当您使用文本-模式流时，将在 C 语言的换行符（\n）和操作系统用于在磁盘文件中标记行尾的字符之间进行转换。在 DOS 系统或 Microsoft Windows 的控制台中，标记行尾的字符是回车符和换行符（CR-LF）。将数据写入到文本-模式文件中时，所有的\n 都被转换为 CR-LF；而读取磁盘文件中的数据时，所有的 CR-LF 都被转换为\n。在 UNIX 系统中，不进行转换：换行符保留不变。

二进制流与二进制-模式文件相关联。读写文件时，数据保留不变，无需将行分隔开来，也不使用行尾字符。NULL 和行尾字符没有特殊含义，处理时与其他数据字节一样。

有些文件输入/输出函数只能用于一种文件模式，而其他函数可用于任何一种模式。今天的课程将介绍各个函数能用于的模式。

www.plcworld.cn

16.3 文件名

每个磁盘文件都有名称，处理磁盘文件时，必须使用文件名。文件名被存储为字符串，就像其他文本数据一样。文件名规则随操作系统而异。在 DOS 和 Windows 3.x 中，完整的文件名由 1～8 个字符的名称、可选的句点和 1～3 个字符的扩展名组成。而在 Windows 95 和更高的版本（包括 NT、XP 和.NET）以及 UNIX 系统中，文件名最多可以包含 256 个字符。

另外，文件名中可以包含的字符也随操作系统而异。例如，在 Windows 95/98 中，文件名不能使用下述字符：

```
/ \ : * ? ' < > |
```

您必须知道您编写的程序将在什么操作系统下运行，并了解这种操作系统的文件名规则。

在 C 程序中，文件名也可以包含路径信息。路径指的是文件所在的驱动器和/或目录。如果您指定文件名时没有提供路径，则系统会认为该文件位于操作系统默认指定的当前目录中。一种好的编程习惯是，总是在文件名中指定路径信息。



> 提示：如果不指定路径，则该文件应该与程序位于同一个目录中。您可以通过编程逻辑来包含当前程序的路径。

在 PC 上，使用反斜杠将路径中的目录隔开。例如，在 DOS 和 Microsoft Windows 中，下面的名称：

```
c:\data\list.txt
```

指的是 C 盘中\data 目录下的 list.txt 文件。在 C 语言中，被用于字符串中时，反斜杠有特殊的含义。要表示反斜杠本身，必须在前面再加上一个反斜杠。因此，在 C 程序中，应这样表示文件名：

```
char *filename = "c:\\data\\list.txt";
```

然而，在运行程序期间使用键盘输入文件名时，只需输入一个反斜杠即可。

并非所有的操作系统都使用反斜杠来分隔目录。例如，UNIX 使用斜杠来分隔。

16.4 打开文件

创建与磁盘文件相关联的流被称为打开文件。文件被打开后，便可以对它进行读（将文件中的数据输入到程序中）写（将数据保存到文件中）。使用完文件后，必须关闭它。关闭文件将在本章后面介绍。

使用库函数 fopen()来打开文件，该函数的原型如下，它是在头文件 stdio.h 中定义的：

```
FILE *fopen(const char *filename, const char *mode);
```

上述原型指出，fopen()返回一个 FILE 指针，FILE 是 stdio.h 声明的一种结构。FILE 结构的成员被程序用来执行各种文件存取操作，但您无需了解它。然而，对于每个要打开的文件，您都必须声明一个 FILE 指针。当您调用 fopen()时，该函数将创建一个 FILE 结构的实例，并返回一个指向该实例的指针。接下来对文件执行的所有操作都是通过该指针来完成的。如果 fopen()运行失败，则返回一个 NULL。失败的原因可能是硬件错误、磁盘未格式化等。

参数 filename 是要打开的文件的名称。正如前面指出的，filename 可以（也应该）包含路径。filename 参数可以是一个用双引号括起的字面字符串，也可以是一个指向字符串的指针。

参数 mode 指定以何种模式打开文件。就这里而言，mode 控制文件是二进制的还是文本的，是读文件、写文件还是读写文件。mode 可能的值如表 16.1 所示。

www.plcworld.cn

表 16.1　　在 fopen()函数中，mode 的可能取值

模　式	含　义
r	打开文件，进行读取。如果文件不存在，fopen()返回 NULL。
w	打开文件，进行写入。如果指定的文件不存在，则创建它；如果指定的文件存在，则删除它（而不发出任何警告），并创建一个新的空文件。
a	打开文件，追加内容。如果指定的文件不存在，则创建它；如果存在，则将新数据追加到文件末尾。
r+	打开文件，进行读写。如果指定的文件不存在，则创建它；如果存在，则将新数据写入到文件开头，覆盖原来的数据。
w+	打开文件，进行读写。如果指定的文件不存在，则创建它；如果存在，则覆盖它。
a+	打开文件，进行读取和追加。如果指定的文件不存在，则创建它；如果存在，则将新数据追加到文件末尾。

默认的文件模式为文本。要以二进制模式打开文件，可在模式参数后加上 b。因此，模式参数 a 表示打开一个文本模式文件，进行追加；而 ab 表示打开一个二进制模式文件，进行追加。

前面指出过，如果发生错误，fopen()将返回 NULL。导致错误的原因有：

- 文件名无效；
- 试图打开未准备好的磁盘（例如，驱动门没有关好、磁盘未格式化）上的文件；
- 试图打开不存在的目录或磁盘驱动器中的文件；
- 试图以 r 模式打开一个不存在的文件。

使用 fopen()时，一定要检查是否发生了错误。您无法知道发生的错误到底是什么，但可以给用户显示一条消息并再次尝试打开它或结束程序。大多数 C 编译器都包含非 ANSI 扩展，使您能够获得有关错误性质的信息，具体的情况请参阅编译器文档。

程序清单 16.1 演示了如何使用 fopen()。

程序清单 16.1　　fopen.c：使用 fopen()以各种模式打开磁盘文件

```
1:  /* Demonstrates the fopen() function. */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:     FILE *fp;
8:     char ch, filename[40], mode[4];
9:
10:    while (1)
11:    {
12:
13:       /* Input filename and mode. */
14:
15:       printf("\nEnter a filename: ");
16:       gets(filename);
17:       printf("\nEnter a mode (max 3 characters): ");
18:       gets(mode);
19:
20:       /* Try to open the file. */
21:
22:       if ( (fp = fopen( filename, mode )) != NULL )
23:       {
24:          printf("\nSuccessful opening %s in mode %s.\n",
25:                filename, mode);
```

www.plcworld.cn


```
26:            fclose(fp);
27:            puts("Enter x to exit, any other to continue.");
28:            if ( (ch = getc(stdin)) == 'x')
29:               break;
30:            else
31:               continue;
32:         }
33:         else
34:         {
35:            fprintf(stderr, "\nError opening file %s in mode %s.\n",
36:                  filename, mode);
37:            puts('Enter x to exit, any other to try again.");
38:            if ( (ch = getc(stdin)) == 'x')
39:               break;
40:            else
41:               continue;
42:         }
43:    }
44: return0;
45:}
```

该程序的运行情况如下：

```
Enter a filename: junk.txt

Enter a mode (max 3 characters): w

Successful opening junk.txt in mode w.
Enter x to exit, any other to continue.
j

Enter a filename: morejunk.txt

Enter a mode (max 3 characters): r

Error opening morejunk.txt in mode r.
Enter x to exit, any other to try again.
x
```

分析：该程序的第15～18行提示用户输入文件名并指定模式。获得文件名后，第22行尝试打开该文件，并将指向该文件的指针赋给fp。第22行的if语句检查返回的指针是否为NULL，这是一种很好的编程习惯。如果fp不为NULL，则打印一条消息，指出已经成功地打开文件，并让用户做出选择，是继续还是退出。如果文件指针为NULL，则执行if语句的else部分（第33～42行），它打印一条消息，指出发生了错误，然后让用户选择是继续还是退出。

您可以指定各种文件名和模式，看看哪些情况下会发生错误。从上述输出可知，试图以r模式打开文件morejunk.txt将发生错误，因为该文件不存在。发生错误后，系统让用户选择是继续输入信息还是退出程序。要使之发生错误，可以输入一个无效的文件名，如[]。

16.5 读写文件数据

使用磁盘文件的程序可以将数据写入文件、读取文件中的数据或兼而有之。可以以三种方式将数据写入

www.plcworld.cn

到文件中：

- 可以使用格式化输出将格式化数据保存到文件中。格式化输出只能用于文本文件。格式化输出主要用于创建包含文本和数字数据的文件，供其他程序（如电子表格或数据库）读取。您很少会使用格式化输出来创建供 C 程序读取的文件。
- 可以使用字符输出将单个的字符或成行的字符保存到文件中。虽然从技术上说，可以将字符输出保存到二进制文件中，但很复杂；因此应仅限于将字符输出保存到文本文件中。字符输出的主要用于以 C 程序或其他程序（如字处理器）能够读取的格式保存文本（不包含数字）数据。
- 可以通过直接输出（direct output）将内存中的内容直接保存到磁盘文件中。这种方法只适用于二进制文件。对于保存数据供 C 程序使用来说，直接输出是最佳的方式。

读取文件中的数据时，也有三种方式可供选择：格式化输入、字符输入和直接输入。使用何种输入类型完全取决于文件的性质。通常，读取数据时采用的模式与保存时相同，但不一定非得这样。然而，要以不同于写入的模式读取文件中的数据，必须对 C 语言和文件格式有深入的了解。

前面描述三种文件输入和输出时，指出了它们各自最适合用于完成的任务。但这些规则绝对不是不可愈越的。C 语言非常灵活（这是它的优点之一），因此聪明的程序员能够使任何一种文件输出适合各种需求。作为一名初级程序员，您可能发现，遵循这些规则，工作起来将更容易（至少开始是这样的）。

16.5.1　格式化文件输入/输出

格式化文件输入/输出用于处理以特定的方式格式化后的文本和数值数据。这与使用 printf()和 scanf()函数格式化键盘输入和屏幕输出（参见第 14 天的课程）类似。下面首先讨论格式化输出，然后讨论格式化输入。

1．格式化文件输出

格式化文件输出是通过库函数 fprintf()来完成的，该函数的原型如下，它位于头文件 stdio.h 中：

```
int fprintf(FILE *fp, char *fmt, ...);
```

第一个参数是一个 FILE 指针。要将数据写入到某个文件中，可以使用 fopen()打开该文件，并将返回的指针传递给 fprintf()。

第二个参数是格式化字符串。第 14 天的课程介绍了 printf()使用的格式化字符串，fprintf()的格式化字符串遵循相同的规则。有关细节，请参阅第 14 天的课程。

最后一个参数是...，这是什么意思呢？在函数原型中，省略号表示可变数目的参数。换句话说，除了文件指针和格式化字符串外，fprintf()还可以接受 0 个、1 个或多个参数，这与 printf()相同。这些参数是要将其输出到指定流中的变量的名称。

fprintf()的工作原理与 printf()类似，只是将输出发送到指定的流中。实际上，如果将流指定为 stdout，则 fprintf()与 printf()完全相同。

程序清单 16.2 演示了 fprintf()的用法。

程序清单 16.2　　fpirntf.c：使用 fprintf()将格式化输出写入到文件和 stdout 中

```
1:  /* Demonstrates the fprintf() function. */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:
5:  void clear_kb(void);
6:
7:  int main( void )
8:  {
9:      FILE *fp;
10:     float data[5];
11:     int count;
```

www.plcworld.cn


```
12:    char filename[20];
13:
14:    puts("Enter 5 floating-point numerical values.");
15:
16:    for (count = 0; count < 5; count++)
17:       scanf("%f", &data[count]);
18:
19:    /* Get the filename and open the file. First clear stdin */
20:    /* of any extra characters. */
21:
22:    clear_kb();
23:
24:    puts("Enter a name for the file.");
25:    gets(filename);
26:
27:    if ( (fp = fopen(filename, "w")) == NULL)
28:    {
29:       fprintf(stderr, "Error opening file %s.", filename);
30:       exit(1);
31:    }
32:
33:    /* Write the numerical data to the file and to stdout. */
34:
35:    for (count = 0; count < 5; count++)
36:    {
37:       fprintf(fp, "\ndata[%d] = %f", count, data[count]);
38:       fprintf(stdout, "\rdata[%d] = %f", count, data[count]);
39:    }
40:    fclose(fp);
41:    printf("\n");
42:    return 0;
43: }
44:
45: void clear_kb(void)
46: /* Clears stdin of any waiting characters. */
47: {
48:    char junk[80];
49:    gets(junk);
50: }
```

该程序的运行情况如下：

```
Enter 5 floating-point numerical values.
3.14159
9.99
1.50
3.
1000.0001
Enter a name for the file.
numbers.txt

data[0] = 3.141590
data[1] = 9.990000
```

www.plcworld.cn


```
data[2] = 1.500000
data[3] = 3.000000
data[4] = 1000.000122
```

您可能感到奇怪，输入的值 1000.0001 为何被显示为 1000.000122。这并非程序错误，而是由于 C 语言在内部存储数字的方式引起的。有些浮点数无法准确地被存储，因此有时会导致很小的误差。

该程序的第 37 和 38 行使用 fprintf()将一些格式化文本和数值数据发送给 stdout 和您指定的文件中。这两行之间唯一不同的是第一个参数——数据将被发送到的流。运行该程序后，请使用编辑器查看文件 numbers.txt 的内容，该文件位于程序文件所在的目录中。您会发现该文件中的文本与屏幕上显示的文本完全相同。

该程序使用了第 14 天课程中的 clear_kb()函数。这是为了删除调用 scanf()时留在 stdin 中的多余字符。如果不清除 stdin，读取文件名的 gets()函数将读取多余的字符（即换行符），从而导致创建文件时发生错误。

2. 格式化文件输入

要读取格式化文件输入，可使用库函数 fscanf()，该函数的用法与 scanf()（参见第 14 天的课程）类似，只是输入来自指定的流，而不是 stdin。fscanf()的原型如下：

```
int fscanf(FILE *fp, const char *fmt, ...);
```

其中参数 fp 是 fopen()返回的一个 FILE 指针，fmt 是一个指向格式化字符串（指定 fscanf()如何读取输入）的指针。格式化字符串的组成与 scanf()的格式化字符串相同。最后，省略号（...）表明其他的一个或多个参数——变量的地址，fscanf()将输入赋给这些变量。

开始使用 fscanf()之前，请复习第 14 天课程中关于 scanf()的一节。函数 fscanf()的工作原理与 scanf()相同，只是它从指定的流（而不是 stdin）中读取字符。

为演示 fscanf()的用法，必须创建一个文本文件，该文件以 fscanf()能够读取的格式保存了一些数字或字符串。使用编辑器创建一个名为 INPUT.TXT 的文件，然后在其中输入 5 个浮点数，并在它们之间加上一些空白（空格或换行符）。例如，该文件的内容可能如下：

```
123.45    87.001
100.02
0.00456   1.0005
```

现在编译并运行程序清单 16.3。

程序清单 16.3　　fscanf.c：使用 fscanf()来读取文件中的格式化数据

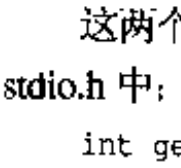
```
1:  /* Reading formatted file data with fscanf(). */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:     float f1, f2, f3, f4, f5;
8:     FILE *fp;
9:
10:    if ( (fp = fopen("INPUT.TXT", "r")) == NULL)
11:       {
12:       fprintf(stderr, "Error opening file.\n");
13:       exit(1);
14:       }
15:
16:    fscanf(fp, "%f %f %f %f %f", &f1, &f2, &f3, &f4, &f5);
17:    printf("The values are %f, %f, %f, %f, and %f\n.",
18:           f1, f2, f3, f4, f5);
```

www.plcworld.cn


```
19:
20:     fclose(fp);
21:     return 0;
22: }
```

该程序的输出如下：

```
The values are 123.449997, 87.000999, 100.019997, 0.004560, and 1.000500
```



注意：值的精度可能使得显示的与实际的不同。例如，100.02 可能被显示为 100.01999。

分析：该程序从您创建的文件中读取 5 个值，然后将它们显示到屏幕上。第 10 行调用 fopen()以读模式打开文件，并检查是否成功。如果没有成功，则显示一条消息（第 12 行），然后退出程序（第 13 行）。第 16 行演示了 fscanf()函数的用法。除了第一个参数外，fscanf()与本书一直在使用的 scanf()完全相同。第一个参数指向要读取的文件。您可以使用编辑器创建其他的输入文件，并试验 fscanf()，以了解 fscanf()是如何读取数据的。

16.5.2 字符输入/输出

涉及磁盘文件时，字符输入/输出指的是单个字符和成行的字符。一行由 0 个或更多的字符组成，以换行符结尾。字符输入/输出用于文本文件。接下来的各节介绍各种字符输入/输出函数，最后给出了一个演示程序。

1．字符输入

用于读取文件的字符输入函数有三个：getc()、fgetc()和 fgets()，前两个函数读取一个字符，最后一个函数读取一行。

（1）getc()和 fgetc()函数

这两个函数相同，可以相互替换。它们从指定的流中读取一个字符。getc()的原型如下，它位于头文件 stdio.h 中：

```
int getc(FILE *fp);
```

参数 fp 是 fopen()打开文件时返回的指针。getc()函数返回读取的字符；如果发生错误，则返回 EOF。

以前的程序使用过 getc()来从键盘读取一个字符。这再次说明了流的灵活性——同一个函数可用于从键盘或文件读取输入。

既然 getc()和 fgetc()返回一个字符，为何它们的原型的返回类型为 int？原因在于，读取文件时，必须能够读取文件尾标记，而在有些系统中，这种标记的类型为 int 而不是 char。程序清单 16.10 演示了如何使用 getc()。



注意：getchar()函数也用于读取字符，但它只能读取 stdin 流中的字符，而不能读取文件中的字符。

（2）fgets()函数

要从文件中读取一行字符，可以使用库函数 fgets()。该函数的原型如下：

```
char *fgets(char *str, int n, FILE *fp);
```

其中参数 str 是指向缓冲区的指针，输入被存储在该缓冲区中；n 是要读取的最大字符数；fp 是一个 FILE 指针，是由 fopen()打开文件时返回的。

被调用后，fgets()将 fp 中的字符读入到（从 str 指向的位置开始的）内存中。然后不断读取字符，直到遇到换行符或读取了 n-1 个字符为止。通过将 n 设置为缓冲区 str 包含的字节数，可以防止输入覆盖缓冲区之外的空间（读取 n-1 个字符，这样为 fgets()在字符串末尾加上的空字符留出了空间）。如果成功，fgets()返回

www.plcworld.cn

str；如果发生错误，则返回 NULL。可能发生的错误有两种：

- 如果还没有将任何字符赋给 str 之前，就发生了读取错误或遇到了 EOF，则返回 NULL，而 str 指向的内存中包含的内容不变；
- 如果发生读取错误或遇到了 EOF 时，至少已经将一个字符赋给 str，则返回 NULL，而 str 指向的内存中包含的是垃圾内容。

fgets()并不一定读取整行（即下一个换行符之前的所有字符）。如果到达换行符之前已经读取了 n-1 个字符，则 fgets()将结束运行。下一次读取该文件时，将从前一次停止的地方开始。为确保 fgets()读取整个字符串——仅在遇到换行符后停止，应确保输入缓冲区和传递给 fgets()的 n 值足够大。

2. 字符输出

您必须了解一些字符输出函数：putc()、fputc()、puts()和 fputs()。

（1）putc()和 fputc()函数

库函数 putc()和 fputc()将一个字符写入到指定的流中。putc()函数的原型如下，它位于 stdio.h 中：

```
int putc(int ch, FILE *fp);
```

参数 ch 是要输出的字符。与其他字符函数一样，该参数的类型也是 int，但只有低端的一个字节被使用。参数 fp 是一个与文件相关联的指针（该指针是由 fopen()打开文件时返回的）。如果成功，函数 putc()返回刚写入的字符；如果发生错误，则返回 EOF。符号常量 EOF 是在 stdio.h 中定义的，它的值为-1。由于任何字符的 ASCII 码都不为-1，因此 EOF 可用作错误指示器（仅限于文本文件）。



注意：putchar()函数也可用来写入字符，但它只能写 stdout，而不能写文件。

（2）fputs()函数

要将一行字符写入到流中，可使用库函数 fputs()。该函数的工作原理与第 14 天课程中介绍的 puts()相同。唯一的差别是，使用 fputs()时，可以指定输出流。另外，fputs()不会在字符串末尾加上换行符；如果需要，您必须明确地包含它。该函数的原型如下，它位于 stdio.h 中：

```
char fputs(char *str, FILE *fp);
```

其中 str 是一个指针，指向一个以空字符结尾的字符串；fp 是一个 FILE 指针，是由 fopen()打开文件时返回的。str 指向的字符串将被写入到文件中，但结尾的空字符除外。如果成功，函数 fputs()返回一个非负值；如果发生错误，则返回 EOF。

16.5.3　直接文件输入/输出

直接文件输入/输出最常见的用途是，保存供 C 程序读取的数据。直接输入/输出只能用于二进制文件。直接输出将内存中的数据块写入到磁盘中；直接输入则相反：将磁盘中的一个数据块读入到内存中。例如，调用一次直接输出函数便可以将整个 double 数组写入到磁盘中；而调用一次直接输入函数便可以将整个数组从磁盘读入到内存中。直接输入/输出函数包括 fread()和 fwrite()。

1. fwrite()函数

库函数 fwrite()将内存中的一个数据块写入到二进制文件中，该函数的原型如下，它位于 stdio.h 中：

```
int fwrite(void *buf, int size, int count, FILE *fp);
```

其中参数 buf 是一个指向内存区域的指针，该内存区域中存储了要写入到文件中的数据。该指针的类型为 void，可以指向任何东西。

参数 size 指定了每个数据项的长度（单位为字节），而 count 指定要写入到磁盘的数据项数。例如，如果要保存一个包含 100 个元素的 int 数组，则 size 应为 2（因为每个 int 占用 2 个字节），而 count 应为 100（因为该数组包含 100 个元素）。要确定参数 size 的值，可以使用 sizeof()运算符。

www.plcworld.cn

参数 fp 是一个 FILE 指针，是由 fopen()在打开文件时返回的。fwrite()函数返回成功写入磁盘的数据项数，如果返回值小于 count，则说明发生了错误。为检查错误，通常在程序中这样使用 fwrite()：

```
if ( fwrite(buf, size, count, fp)) != count)
fprintf(stderr, "Error writing to file.");
```

下面是一些使用 fwrite()的范例。要将 double 变量 x 写入到文件中，可以这样编写代码：

```
fwrite(&x, sizeof(double), 1, fp);
```

要将数组 data[]（它包含 50 个元素，其中每个元素都是一个 address 结构）写入到文件中，有两种方式：

```
fwrite(data, sizeof(address), 50, fp);
fwrite(data, sizeof(data), 1, fp);
```

前一种方式将数组作为 50 个元素写入到文件中，其中每个元素的长度为 address 结构的长度；第二种方法将整个数组作为一个整体写入到文件中。这两种方法完成的工作完全相同。

接下来的一节介绍 fread()，然后给出一个演示 fread()和 fwrite()的程序。

2. fread()函数

库函数 fread()将二进制文件中的一个数据块读入到内存中。该函数的原型如下，它位于 stdio.h 中：

```
int fread(void *buf, int size, int count, FILE *fp);
```

其中参数 buf 是一个指向内存区域的指针，该内存区域将用于存储从文件中读取的数据。与 fwrite()一样，该指针的类型也是 void。

参数 size 指定了要读取的每个数据项的长度（单位为字节），而 count 指定要读取的数据项数。请注意这些参数与 fwrite()使用的参数之间的相似性。同样，可以使用 sizeof()运算符来提供 size 参数。参数 fp 是一个 FILE 指针，是由 fopen()在打开文件时返回的。fread()函数返回成功读取的数据项数，如果达到文件尾或发生错误，返回值将小于 count。

程序清单 16.4 演示了 fwrite()和 fread()的用法。

程序清单 16.4　　direct.c：使用 fwrite()和 fread()直接存取文件

```
1:  /* Direct file I/O with fwrite() and fread(). */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:
5:  #define SIZE 20
6:
7:  int main( void )
8:  {
9:     int count, array1[SIZE], array2[SIZE];
10:    FILE *fp;
11:
12:    /* Initialize array1[]. */
13:
14:    for (count = 0; count < SIZE; count++)
15:       array1[count] = 2 * count;
16:
17:    /* Open a binary mode file. */
18:
19:    if ( (fp = fopen("direct.txt", "wb")) == NULL)
20:    {
21:       fprintf(stderr, "Error opening file.");
22:       exit(1);
23:    }
24:    /* Save array1[] to the file. */
```

www.plcworld.cn


```
25:
26:     if (fwrite(array1, sizeof(int), SIZE, fp) != SIZE)
27:     {
28:         fprintf(stderr, "Error writing to file.");
29:         exit(1);
30:     }
31:
32:     fclose(fp);
33:
34:     /* Now open the same file for reading in binary mode. */
35:
36:     if ( (fp = fopen("direct.txt", "rb")) == NULL)
37:     {
38:         fprintf(stderr, "Error opening file.");
39:         exit(1);
40:     }
41:
42:     /* Read the data into array2[]. */
43:
44:     if (fread(array2, sizeof(int), SIZE, fp) != SIZE)
45:     {
46:         fprintf(stderr, "Error reading file.");
47:         exit(1);
48:     }
49:
50:     fclose(fp);
51:
52:     /* Now display both arrays to show they're the same.*/
53:
54:     for (count = 0; count < SIZE; count++)
55:         printf("%d\t%d\n", array1[count], array2[count]);
56:     return 0;
57: }
```

该程序的输出如下：

```
0       0
2       2
4       4
6       6
8       8
10      10
12      12
14      14
16      16
18      18
20      20
22      22
24      24
26      26
28      28
30      30
32      32
```

www.plcworld.cn


```
34      34
36      36
38      38
```

分析：程序清单演示了函数 fwrite()和 fread()的用法。该程序的第 14 和 15 行初始化一个数组，然后在第 26 行使用 fwrite()将该数组保存到磁盘中。程序的第 44 行使用 fread()将数据读取到另一个数组中。最后，将这两个数组显示到屏幕上，表明它们包含的数据相同（第 54 和 55 行）。

使用 fwrite()保存数据时，除了某些磁盘错误外，犯其他错误的可能性很小。然而，使用 fread()时一定要小心。在 fread()看来，磁盘上的数据就是一个字节序列，该函数并不知道它表示的是什么。例如，在 16 位的系统上，一个 100 字节的数据块可能包含 100 个 char 变量、50 个 int 变量、25 个 long 变量或 25 个 float 变量。如果您命令 fread()将该数据块读入到内存中，它可能这样做。然而，如果数据块中存储的是一个 int 数组，而您将它读入到一个 float 数组中，将不会发生错误，但结果将是怪异的。编写程序时，必须正确地使用 fread()——将数据读取到类型正确的变量和数组中。在程序清单 16.4 中，调用 fopen()、fwrite()和 fread()时，都进行了检查，以确保它们正确地工作。

16.6 文件缓冲技术：关闭和刷新文件

使用完文件后，应使用 fclose()函数将其关闭。今天课程中前面的程序已经使用过 fclose()函数。该函数的原型如下：

```
int fclose(FILE *fp);
```

其中参数 fp 是一个与流相关联的 FILE 指针。如果成功，fclose()返回 0；否则返回-1。关闭文件时，将刷新文件的缓冲区（写入到文件中）。也可以使用 fcolseall()来关闭除标准流（stdin、stdout、stderr、stdprn 和 stdaux）之外的所有打开的流。该函数的原型如下：

```
int fcloseall(viod);
```

该函数刷新所有的流缓冲区，并返回关闭的流数。

程序结束时（无论是由于达到 main()的末尾，还是由于执行了 exit()函数），所有流都将自动被刷新和关闭。然而，使用完毕后，立刻显式地关闭流（尤其是那些与文件相关联的流）是个不错的主意。原因与流缓冲区相关。

创建与磁盘文件相关联的流时，会自动创建一个缓冲区，并将其与流关联起来。缓冲区是一个内存块，用于临时存储被写入到文件和从文件读取的数据。之所以需要缓冲区是因为磁盘驱动器是一种面向块的设备，即以特定长度的块读写数据时，它们的效率最高。理想的块长随硬件而异，通常为几百个字节。然而，您无需关心块长到底是多少。

与文件流相关联的缓冲区充当了流（它是面向字符的）和磁盘硬件（是面向块的）之间的桥梁。程序将数据写入流时，数据首先被保存到缓冲区中，缓冲区被填满后，整个缓冲区的内容将以块的方式被写入磁盘。读取磁盘文件中的数据的过程与此类似。创建和操纵缓冲区的工作是由操作系统完成的，这完全是自动进行的，您无需担心（C 语言确实提供了一些操纵缓冲区的函数，但这已超出了本书的范围）。

实际上，缓冲区的这种运行方式意味着在程序执行期间，程序写入到磁盘的数据仍留在缓冲区，而没有进入磁盘。如果程序被挂起、突然断电或者发生其他问题，留在缓冲区中的数据可能丢失，而您将无法知道磁盘文件中包含的内容。

可以在不关闭流的情况下，使用库函数 fflush()或 flushall()来刷新流的缓冲区。当需要继续使用文件，同时想将文件缓冲区中的数据写入到磁盘时，可使用 fflush()；当需要刷新所有打开的流的缓冲区时，可以使用 flushall()。这两个函数的原型如下：

```
int fflush(FILE *fp);
int flushall(void);
```

其中参数 fp 是一个 FILE 指针，是 fopen()在打开文件时返回的。如果文件是为写入而打开的，则 fflush()将其缓冲区中的数据写入磁盘；如果文件是为读取而打开的，则 fflush()清除其缓冲区中的内容。成功时，函

www.plcworld.cn

数 fflush()返回 0；如果发生错误，则返回 EOF。函数 flushall()返回处于打开状态的流的数目。

应　该	不 应 该
读写文件之前，一定要打开它； 使用函数 fwrite()和 fread()时，应使用 sizeof()运算符来确定数据项的长度； 一定要关闭您打开的所有文件。	不要假设存取文件是成功的。读、写、打开文件时，一定要进行检查，以确保函数已成功地执行。 除非要关闭所有的流，否则不要使用 fcloseall()函数。

16.7　顺序文件存取和随机文件存取

每个打开的文件都有一个相应的文件位置指示器。位置指示器指定在文件的什么位置执行读写操作。位置总是以偏离文件开头的字节数来表示。新文件被打开时，位置指示器总是指向文件的开头——位置 0（由于文件是新的，其长度为 0，因此没有其他的位置可言）。以追加模式打开已有的文件时，位置指示器指向文件末尾；以其他模式打开已有的文件时，位置指示器指向文件的开头。

本章前面介绍的文件输入/输出函数使用了位置指示器，虽然操纵工作是在幕后进行的。读写操作是在位置指示器指向的位置进行的，同时会更新位置指示器。例如，如果您打开一个文件进行读取，并读取了 10 个字节，则输入的是文件的开头 10 个字节（位置 0~9 处的字节）。读取操作完成后，位置指示器指向位置 10，下一次读取操作将从这里开始。因此，如果要顺序地读取文件中的所有数据或者要将数据顺序地写入文件，并不需要关心位置指示器。流输入/输出函数会自动处理位置指示器。

当您需要有更大的控制权时，可以使用能够让您确定和修改文件位置指示器的库函数。通过控制位置指示器，可以进行随机文件存取。在这里，随机指的是可以在文件的任何位置读写数据，而不必先读写前面的所有数据。

16.7.1　ftell()和 rewind()函数

要让位置指示器指向文件开头，可以使用库函数 rewind()。该函数的原型如下，它位于 stdio.h 中：

```
void rewind(FILE *fp);
```

其中参数 fp 是一个与流相关联的 FILE 指针。调用 rewind()后，文件的位置指示器将指向文件开头（字节 0）。如果您已经读取了文件中的一些数据，并想再次从文件开头进行读取，而又不想关闭并重新打开该文件，则可以使用 rewind()函数。

要确定文件的位置指示器的值，可以使用 ftell()。该函数的原型如下，它位于 stdio.h 中：

```
long ftell(FILE *fp);
```

其中参数 fp 是 fopen()打开文件时返回的一个 FILE 指针。函数 ftell()返回一个 long 值，指出当前位置离文件开头有多少个字节（文件第一个字节的位置为 0）。如果发生错误，ftell()返回-1L（类型为 long 的-1）。

程序清单 16.5 演示了如何使用 rewind()和 ftell()函数。

程序清单 16.5　　ftell.c：使用 rewind()和 ftell()

```
1:  /* Demonstrates ftell() and rewind(). */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:
5:  #define BUFLEN 6
6:
7:  char msg[] = "abcdefghijklmnopqrstuvwxyz";
8:
9:  int main( void )
10: {
```

www.plcworld.cn


```
11:     FILE *fp;
12:     char buf[BUFLEN];
13:
14:     if ( (fp = fopen("TEXT.TXT", "w")) == NULL)
15:     {
16:        fprintf(stderr, "Error opening file.");
17:        exit(1);
18:     }
19:
20:     if (fputs(msg, fp) == EOF)
21:     {
22:        fprintf(stderr, "Error writing to file.");
23:        exit(1);
24:     }
25:
26:     fclose(fp);
27:
28:     /* Now open the file for reading. */
29:
30:     if ( (fp = fopen("TEXT.TXT", "r")) =- NULL)
31:     {
32:        fprintf(stderr, "Error opening file.");
33:        exit(1);
34:     }
35:     printf("\nImmediately after opening, position = %ld", ftell(fp));
36:
37:     /* Read in 5 characters. */
38:
39:     fgets(buf, BUFLEN, fp);
40:     printf("\nAfter reading in %s, position = %ld", buf, ftell(fp));
41:
42:     /* Read in the next 5 characters. */
43:
44:     fgets(buf, BUFLEN, fp);
45:     printf("\n\nThe next 5 characters are %s, and position now = %ld",
46:           buf, ftell(fp));
47:
48:     /* Rewind the stream. */
49:
50:     rewind(fp);
51:
52:     printf("\n\nAfter rewinding, the position is back at %ld",
53:           ftell(fp));
54:
55:     /* Read in 5 characters. */
56:
57:     fgets(buf, BUFLEN, fp);
58:     printf("\nand reading starts at the beginning again: %s\n", buf);
59:     fclose(fp);
60:     return 0;
61: }
```

www.plcworld.cn

该程序的输出如下：

```
Immediately after opening, position = 0
After reading in abcde, position = 5

The next 5 characters are fghij, and position now = 10

After rewinding, the position is back at 0
and reading starts at the beginning again: abcde
```

分析：该程序将一个名为 msg 的字符串写入到文件 text.txt 中。该字符串包含 26 个字母（按字母顺序排列）。第 14～18 行以写模式打开文件 text.txt，然后确保文件被成功地打开。第 20～24 行使用 fputs()将 msg 写入到文件中，并确保已经成功地写入。第 26 行使用 fclose()关闭该文件。至此，便创建了一个供程序其他部分使用的文件。

第 30～34 行以读模式再次打开该文件。第 35 行打印 ftell()返回的值，此时位于文件的开头。第 39 行使用 gets()来读取 5 个字符，而第 40 行打印读取的 5 个字符和新的文件位置。ftell()返回的偏离值是正确的。第 50 行调用 rewind()将指针重新指向文件的开始位置，然后第 52 行再次打印文件位置。这表明，rewind()确实是将指示器重新指向文件开头。第 57 行再次进行读取，以验证指示器确实是指向文件的开头。第 59 行在程序结束之前将文件关闭。

16.7.2　fseek()函数

要更精确地控制流的位置指示器，可以使用库函数 fseek()。使用 fseek()可以将位置指示器指向文件的任何位置。该函数的原型如下，它位于 stdio.h 中：

```
int fseek(FILE *fp, long offset, int origin);
```

其中参数 fp 是一个与文件相关联的 FILE 指针。指示器移动的距离由 offset 指定（单位为字节）。参数 origin 指定移动距离是相对什么位置的，可能的取值有三个，这三个值是 io.h 中定义的符号常量，如表 16.2 所示。

表 16.2　　fseek()中的 origin 参数的可能取值

常　量	值	描　述
SEEK_SET	0	将指示器移到离文件开头 offset 字节的位置
SEEK_CUR	1	将指示器移到离当前位置 offset 字节的位置
SEEK_END	2	将指示器移到离文件末尾 offset 字节的位置

如果成功地移动指示器，则 fseek()返回 0；如果发生错误，则返回一个非零值。程序清单 16.6 使用 fseek()来随机存取文件。

程序清单 16.6　　fseek.c：使用 fseek()来随机存取文件

```
1:  /* Random access with fseek(). */
2:
3:   #include <stdlib.h>
4:  #include <stdio.h>
5:
6:  #define MAX 50
7:
8:  int main( void )
9:  {
10:     FILE *fp;
11:     int data, count, array[MAX];
12:     long offset;
13:
```

www.plcworld.cn


```
14:    /* Initialize the array. */
15:
16:    for (count = 0; count < MAX; count++)
17:       array[count] = count * 10;
18:
19:    /* Open a binary file for writing. */
20:
21:    if ( (fp = fopen("RANDOM.DAT", "wb")) == NULL)
22:    {
23:       fprintf(stderr, "\nError opening file.");
24:       exit(1);
25:    }
26:
27:    /* Write the array to the file, then close it. */
28:
29:    if ( (fwrite(array, sizeof(int), MAX, fp)) != MAX)
30:    {
31:       fprintf(stderr, "\nError writing data to file.");
32:       exit(1);
33:    }
34:
35:    fclose(fp);
36:
37:    /* Open the file for reading. */
38:
39:    if ( (fp = fopen("RANDOM.DAT", "rb")) == NULL)
40:    {
41:       fprintf(stderr, "\nError opening file.");
42:       exit(1);
43:    }
44:
45:    /* Ask user which element to read. Input the element */
46:    /* and display it, quitting when -1 is entered. */
47:
48:    while (1)
49:    {
50:       printf("\nEnter element to read, 0-%d, -1 to quit: ",MAX-1);
51:       scanf("%ld", &offset);
52:
53:       if (offset < 0)
54:          break;
55:       else if (offset > MAX-1)
56:          continue;
57:
58:       /* Move the position indicator to the specified element. */
59:
60:       if ( (fseek(fp, (offset*sizeof(int)), SEEK_SET)) != 0)
61:       {
62:          fprintf(stderr, "\nError using fseek().");
63:          exit(1);
64:       }
65:
```

www.plcworld.cn


```
66:        /* Read in a single integer. */
67:
68:        fread(&data, sizeof(int), 1, fp);
69:
70:        printf("\nElement %ld has value %d.", offset, data);
71:    }
72:
73:    fclose(fp);
74:    return 0;
75: }
```

该程序的运行情况如下：

```
Enter element to read, 0-49, -1 to quit: 5

Element 5 has value 50.
Enter element to read, 0 49, -1 to quit: 6

Element 6 has value 60.
Enter element to read, 0-49, -1 to quit: 49

Element 49 has value 490.
Enter element to read, 0-49, -1 to quit: 1

Element 1 has value 10.
Enter element to read, 0-49, -1 to quit: 0

Element 0 has value 0.
Enter element to read, 0-49, -1 to quit: -1
```

分析：第 14～35 行与程序清单 16.5 类似。第 16 和 17 行初始化一个包含 50 个 int 元素的数组 data，其中每个数组元素中存储的值为相应索引的 10 倍。然后将该数组写入到一个名为 RANDOM.DAT 的二进制文件中。您之所以知道这是一个二进制文件，是因为第 21 行打开它时使用的是“wb”模式。

第 39 行再次以二进制读取模式打开该文件。然后是一个 while 死循环，该循环提示用户输入一个数值，以指定要读取多少个数组元素。第 53～56 行检查用户输入的值是否合理。C 语言允许读取文件末尾后面的数据吗？是的，就像允许读取数组末尾后面的数据一样，C 语言也允许您读取文件末尾后面的数据。但如果读取文件末尾后面（或文件开头前面）的数据，结果将是不可预测的。因此最好对您所做的操作进行检查（就像该程序的第 53～56 行那样）。

用户输入要读取的元素后，第 60 行调用 fseek()来跳转到相应的位置。由于使用的是 SEEK_SET，因此相对的是文件开头。注意，移动的距离不是 offset，而是它与元素长度的乘积。然后第 68 和 70 行读取并打印元素。

16.8　检测文件尾

有时候，您已经知道文件的长度，因此不需检测文件尾。例如，如果您使用 fwrite()来保存一个包含 100 个元素的 int 数组，则您知道文件的长度为 200 个字节（假设 int 的长度为 2 字节）。而在其他时候，您并不知道文件的长度，但仍要从文件的开头读取数据，直到文件末尾。检测文件尾的方式有两种。

逐字节地读取文本文件时，可以检测标记文件尾的字符。stdio.h 中定义的符号常量 EOF 的值为-1，任何字符的 ASCII 码都不为-1。当字符输入函数从文本模式流中读取 EOF 时，您便可以断定已到达文件末尾。例

www.plcworld.cn

如，您可以编写下面这样的代码：

```
while ( (c = fgetc( fp )) != EOF )
```

对于二进制模式的流，您不能通过-1 来检测文件尾，因为二进制流中的字节可能包含这样的值，从而导致提早结束输入。不过，您可以使用库函数 feof()来检测文件尾，该函数也可用于文本文件：

```
int feof(FILE *fp);
```

参数 fp 是 fopen()打开文件时返回的一个 FILE 指针。如果没有到达文件 fp 的末尾，则函数 feof()返回 0；否则返回一个非零值。如果 feof()发现已到达文件尾，则不允许再读取文件，直到调用 rewind()和 fseek()函数或者关闭并重新打开该文件为止。

程序清单 16.7 演示了 feof()的用法。当程序提示输入文件名时，请输入一个文本文件（如 C 语言源代码文件或诸如 stdio.h 等头文件）的名称。应确保该文件位于当前目录中，否则应同时输入路径。程序每次读取该文件中的一行，并将其显示到 stdout，直到 feof()检测到文件尾为止。

程序清单 16.7　　feof.c：使用 feof()来检测文件尾

```
1:  /* Detecting end-of-file. */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:
5:  #define BUFSIZE 100
6:
7:  int main( void )
8:  {
9:      char buf[BUFSIZE];
10:     char filename[60];
11:     FILE *fp;
12:
13:     puts("Enter name of text file to display: ");
14:     gets(filename);
15:
16:     /* Open the file for reading. */
17:     if ( (fp = fopen(filename, "r")) == NULL)
18:     {
19:         fprintf(stderr, "Error opening file.");
20:         exit(1);
21:     }
22:
23:     /* If end of file not reached, read a line and display it. */
24:
25:     while ( !feof(fp) )
26:     {
27:         fgets(buf, BUFSIZE, fp);
28:         printf("%s",buf);
29:     }
30:
31:     fclose(fp);
32:     return 0;
33: }
```

该程序的运行情况如下：

```
Enter name of text file to display:
hello.c
```

www.plcworld.cn


```
#include <stdio.h>
int main( void )
{
    printf("Hello, world.");
    return(0);
}
```

分析：该程序中的 while 循环（第 25～29 行）是进行顺序文件存储的程序使用的典型 while 语句。只要还未到达文件尾，while 循环体中的代码（第 27 和 28 行）便不断执行。当 feof()返回一个非零值后，循环将终止，然后程序结束。

应　该	不 应 该
可使用 rewind()或 fseek(fp, SEEK_SET, 0)将文件位置重置到文件开头； 操纵二进制文件时，一定要使用 feof()来检测文件尾。	读取文件时，不要超越文件开头或末尾，可通过检测文件位置来避免这一点； 不要通过 EOF 来检测二进制文件尾。

16.9　文件管理函数

文件管理指的是处理已有的文件——不是读写文件，而是删除、复制和重命名文件。C 标准库中包含了用于删除和重命名文件的函数，而您也可以编写自己的文件复制程序。

16.9.1　删除文件

要删除文件，可以使用库函数 remove()。该函数的原型如下，它位于 stdio.h 中：

```
int remove( const char *filename );
```

其中参数 filename 是一个指针，指向要删除的文件（参见本章前面关于文件名的一节）。指定的文件当前不能处于打开状态。如果指定的文件存在，则将被删除（就像在 DOS 提示符、Windows 命令提示符下执行 DEL 命令或在 UNIX 中执行 rm 命令一样），然后 remove()返回 0。如果指定的文件不存在、是只读的、您没有足够的访问权限或者发生其他错误，则 remove()返回-1。

程序清单 16.8 演示了 remove()的用法。警告：如果您删除一个文件，它将永远消失。

程序清单 16.8　　remove.c：使用 remove()删除磁盘文件

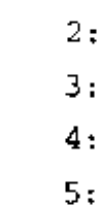
```
1:  /* Demonstrates the remove() function. */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:      char filename[80];
8:
9:      printf("Enter the filename to delete: ");
10:     gets(filename);
11:
12:     if ( remove(filename) == 0)
13:        printf("The file %s has been deleted.\n", filename);
14:     else
15:        fprintf(stderr, "Error deleting the file %s.\n", filename);
16:     return 0;
17: }
```

www.plcworld.cn

该程序的运行情况如下：

```
Enter the filename to delete: *.bak
Error deleting the file *.bak.
Enter the filename to delete: list1414.bak
The file list1414.bak has been deleted.
```

分析：该程序的第 9 行提示用户输入要删除的文件的名称。然后，第 12 行调用 remove()来删除指定的文件。如果返回值为 0，则说明文件已被删除，因此显示一条消息指出这一点。如果返回值不为 0，则说明发生了错误，未能删除文件。



提示：确认用户是否确实要将文件删除总是一个不错的主意。

16.9.2 给文件重命名

函数 rename()用于修改已有磁盘文件的名称。该函数的原型如下，它位于 stdio.h 中：

```
int rename( const char *oldname, const char *newname );
```

参数 oldname 和 newname 遵循本章前面关于文件名的一节介绍的规则。唯一的限制是，两个文件名指定的文件必须位于相同的磁盘驱动器中，您不能给一个文件重命名并将其移到另一个磁盘驱动器中。如果成功，函数 rename()返回 0；如果发生错误，则返回-1。导致错误的原因有：

- 文件 oldname 不存在；
- 文件 newname 已经存在；
- 试图将文件重命名并将其移到另一个磁盘驱动器中。

程序清单 16.9 演示了 rename()的用法。

程序清单 16.9　　renamelt.c：使用 rename()修改磁盘文件的名称

```
1:  /* Using rename() to change a filename. */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:     char oldname[80], newname[80];
8:
9:     printf("Enter current filename: ");
10:    gets(oldname);
11:    printf("Enter new name for file: ");
12:    gets(newname);
13:
14:    if ( rename( oldname, newname ) == 0 )
15:       printf("%s has been renamed %s.\n", oldname, newname);
16:    else
17:       fprintf(stderr, "An error has occurred renaming %s.\n", oldname);
18:    return 0;
19: }
```

该程序的运行情况如下：

www.plcworld.cn


```
Enter current filename: list1609.c
Enter new name for file: renameit.c
list1609.c has been renamed renameit.c.
```

程序清单 16.9 说明了 C 语言的强大功能。该程序只包含 18 行代码，但却可以替代一个操作系统命令，这是一个非常友好的功能。

第 9 行提供用户输入要对其进行重命名的文件。第 11 行提示用户输入新的文件名。第 14 行的 if 语句调用 rename()函数，并检查重命名工作是否成功地完成。如果是，则第 15 行打印一条消息，对此进行确认；否则第 17 行打印一条消息，指出发生了错误。

16.9.3　复制文件

您经常需要制作文件的副本——内容相同但名称不同（或名称也相同，但位于不同的驱动器或目录中）。在 DOS 中，可以使用命令 COPY 来复制，而在其他操作系统中，也有功能相同的命令。在 C 语言中，如何复制文件呢？并没有这样的库函数，因此您必须自己编写。

这看起来有些复杂，但由于 C 语言使用流来进行输入和输出，因此非常简单。请按下面的步骤进行：

1. 以二进制模式打开源文件进行读取（采用二进制模式可以确保该函数能够复制任何文件，而不仅仅是文本文件）；
2. 以二进制模式打开目标文件进行写入；
3. 读取源文件中的一个字符。还记得吗，文件刚打开时，位置指示器指向的是文件开头，因此无需显式地放置指示器；
4. 如果 feof()表明已到达源文件的末尾，则关闭这两个文件，并返回到调用程序；
5. 如果没有到达源文件的末尾，则将该字符写入到目标文件，然后回到第 3 步。

程序清单 16.10 包含一个名为 copy_file()的函数，该函数接受源文件和目标文件的名称，并按前面介绍的步骤执行复制操作。如果打开这两个文件中的任何一个时发生错误，则该函数不进行复制，并将-1 返回给调用程序。复制完毕后，函数关闭这两个文件，并返回 0。

程序清单 16.10　　copyit.c：一个复制文件的函数

```
1:  /* Copying a file. */
2:
3:  #include <stdio.h>
4:
5:  int file_copy( char *oldname, char *newname );
6:
7:  int main( void )
8:  {
9:     char source[80], destination[80];
10:
11:    /* Get the source and destination names. */
12:
13:    printf("\nEnter source file: ");
14:    gets(source);
15:    printf("\nEnter destination file: ");
16:    gets(destination);
17:
18:    if ( file_copy( source, destination ) == 0 )
19:       puts("Copy operation successful");
20:    else
21:       fprintf(stderr, "Error during copy operation");
22:    return 0;
```

www.plcworld.cn


```
23: }
24: int file_copy( char *oldname, char *newname )
25: {
26:    FILE *fold, *fnew;
27:    int c;
28:
29:    /* Open the source file for reading in binary mode. */
30:
31:    if ( ( fold = fopen( oldname, "rb" ) ) == NULL )
32:       return -1;
33:
34:    /* Open the destination file for writing in binary mode. */
35:
36:    if ( ( fnew = fopen( newname, "wb" ) ) == NULL )
37:    {
38:       fclose ( fold );
39:       return -1;
40:    }
41:
42:    /* Read one byte at a time from the source; if end of file */
43:    /* has not been reached, write the byte to the */
44:    /* destination. */
45:
46:    while (1)
47:    {
48:       c = fgetc( fold );
49:
50:       if ( !feof( fold ) )
51:          fputc( c, fnew );
52:       else
53:          break;
54:    }
55:
56:    fclose ( fnew );
57:    fclose ( fold );
58:
59:    return 0;
60: }
```

该程序的运行情况如下：

```
Enter source file: list1610.c

Enter destination file: tmpfile.c
Copy operation successful
```

分析：函数copy_file()能够正常运行，可以复制任何文件——从小型的文本文件到大型的程序文件。然而，它确实有局限性。如果目标文件已经存在，函数将删除它，而不询问用户。您可以修改copy_file()，使之检查目标文件是否已经存在，如果存在，再询问用户是否要覆盖它。

对于程序清单16.10中的main()函数，您应该很熟悉。除第14行外，它几乎与程序清单16.9中的main()函数完全相同。这里要执行的操作是复制，而不是重命名。由于C语言并没有提供复制函数，因此第24～60行创建了一个这样的函数。第31和32行以二进制读取模式打开源文件fold；而第36～40行以二进制写入模式打开目标文件fnew。在第38行中，如果打开目标文件时发生错误，则关闭源文件。第46～54行的while

www.plcworld.cn

循环完成复制文件的工作。第 48 行从源文件 fold 中读取一个字符，然后第 50 行检查读取的是否是文件尾标记。如果已经到达文件尾，则执行 break 语句，跳出 while 循环；否则，将该字符写入到目标文件 fnew 中。返回 main()之前，第 56 和 57 行关闭这两个文件。

16.10　使用临时文件

有些程序在执行期间使用临时文件。临时文件是程序在执行期间为某种用途而创建的，程序结束前，删除临时文件。创建临时文件时，您并不在乎它们的名称，因为它们将被删除。唯一的要求是，名称不能与另一个已有的文件相同。C 标准库中包含一个名为 tmpnam()的函数，可用来创建有效的、且不与任何已有文件冲突的名称。该函数的原型如下，它位于 stdio.h 中：

```
char *tmpnam(char *s);
```

参数 s 是一个指针，它指向的缓冲区必须足够大，能够存储文件名。也可以给函数传递一个空指针（NULL），在这种情况下，临时名被存储在 tmpman()函数内的一个缓冲区中，而该函数将返回一个指向该缓冲区的指针。程序清单 16.11 演示了这两种使用 tmpnam()来创建临时文件名的方法。

程序清单 16.11　　tmpnam.c：使用 tmpnam()创建临时文件名

```
1:  /* Demonstration of temporary filenames. */
2:
3:  #include <stdio.h>
4:
5:  int main( void )
6:  {
7:      char buffer[10], *c;
8:
9:      /* Get a temporary name in the defined buffer. */
10:
11:     tmpnam(buffer);
12:
13:     /* Get another name, this time in the function's */
14:     /* internal buffer. */
15:
16:     c = tmpnam(NULL);
17:
18:     /* Display the names. */
19:     printf("Temporary name 1: %s", buffer);
20:     printf("\nTemporary name 2: %s\n", c);
21: return 0;
22: }
```

该程序的输出如下：

```
Temporary name 1: \s3us.
Temporary name 2: \s3us.1
```



注意：您运行该程序时，创建的临时文件名可能不同。

分析：该程序只是生成并打印临时文件名，而没有创建任何文件。第 11 行将一个临时文件名存储到字符数组 buffer 中。第 16 行将 temnam()返回的字符指针赋给 c。程序必须使用生成的名称来打开临时文件，并在

www.plcworld.cn

终止之前删除该文件。下面的代码片段说明了这一点：

```
char tempname[80];
FILE *tmpfile;
tmpnam(tempname);
tmpfile = fopen(tempname, "w");  /* Use appropriate mode */
fclose(tmpfile);
remove(tempname);
```

应　该	不应该
一定要删除您创建的临时文件，系统并不会自动删除它们。	不要删除您可能还要使用的文件。

16.11　总　结

今天的课程介绍了如何使用磁盘文件。C 语言像处理流（字节序列）那样处理磁盘文件，就像第 14 天课程介绍的预定义流一样。使用之前，必须打开与磁盘文件相关联的流，使用完毕后，必须关闭。可以以文本或二进制模式来打开磁盘文件流。

打开磁盘文件后，便可以将其中的数据读入到内存中，也可以将程序中的数据写入到该文件中。文件输入/输出有三种：格式化输入/输出、字符输入/输出和直接输入/输出。其中每种输入/输出都有最适合完成的数据存储和检索任务。

每个打开的文件都有一个与之相关联的文件位置指示器。指示器指定接下来的读写操作将在文件的什么位置（离文件开头多少个字节）进行。执行文件存取操作时，位置指示器将自动更新，而无需您操心。为实现文件随机存取，C 标准库提供了操纵位置指示器的函数。

最后，C 语言提供了一些基本的文件管理函数，让您能够删除文件、给文件重命名。今天的课程还开发了一个用于复制文件的函数。

16.12　问与答

问：使用诸如 remove()、rename()、fopen()等文件函数时，可以在文件名中指定磁盘驱动器和路径吗？

答：可以。文件名中可以包含路径和磁盘驱动器，也可以不包含。如果没有包含，则函数将在当前目录中查找文件。请记住，使用反斜杠（\）时，需要使用转义序列。另外，UNIX 系统使用斜杠（/）来分隔目录。

问：可以读取文件末尾后面的数据吗？

答：可以。也可以读取文件开头前面的数据。但如果这样做，结果将是灾难性的。读取文件就想操纵数组一样，您必须考虑偏移距离。使用 fseek()时，您应该进行检查，确保位置指示器没有指向文件末尾的后面。

问：如果没有关闭文件，情况将如何？

答：一种好的编程习惯是关闭所有打开的文件。默认情况下，当程序结束时，文件将被关闭；然而，决不要依靠这一点。如果不关闭文件，以后您可能无法存取它，因为操作系统可能认为该文件正被其他程序使用。

问：可以同时打开多少个文件？

答：这取决于操作系统的变量设置。在 DOS 系统中，有一个名为 FILES 的环境变量，用于指定可同时打开多少个文件（这包括正在运行的程序）。更详细的信息请参阅操作系统手册。

问：可以使用随机存取函数顺序地读取文件吗？

答：顺序地读取文件时，无需使用诸如 fseek()这样的函数。因为文件指针将停留在上次读取完毕时的位置，下次读取时总是从这里开始。顺序读取文件时，您可以使用 fseek()函数，但这样做不会有

www.plcworld.cn

任何好处。

16.13　作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

16.13.1　小测验

1. 文本模式流和二进制模式流之间有何区别？
2. 存取磁盘文件前，首先必须如何做？
3. 使用 fopen()打开文件时，必须指定哪些信息？该函数返回什么？
4. 文件存取方法有哪三种？
5. 读取文件的方法有哪两种？
6. EOF 的值是多少？
7. 何时使用 EOF？
8. 在文本和二进制模式下，如何检测文件尾？
9. 文件位置指示器是什么？如何修改它？
10. 文件刚被打开时，文件位置指示器指向什么位置（如果您不确定，请参阅程序清单 16.5）？

16.13.2　练习

1. 编写关闭所有文件流的代码。
2. 使用两种方法将文件位置指针指向文件开头。
3. 排错：下面的代码有什么错误？

```
FILE *fp;
int c;

if ( ( fp = fopen( oldname, "rb" ) ) == NULL )
     return -1;

while (( c = fgetc( fp)) != EOF )
     fprintf( stdout, "%c", c );
fclose ( fp );
```

由于下面的练习有多种解决方案，因此附录 F 没有提供它们的答案。

4. 编写一个程序，将一个文件的内容显示到屏幕上。
5. 编写一个程序，它打开一个文件，并将其内容发送到打印机（stdprn）。打印时，每页最多 55 行。
6. 修改练习 5 中的程序，在每页打印页眉，页眉包含文件名和页码。
7. 编写一个程序，它打开一个文件并计算和打印其中包含的字符数。
8. 编写一个程序，它打开一个已有的文本文件，并将其内容复制到一个新文件中。复制时，将所有的小写字母改为大写，其他字符不变。
9. 编写一个程序，它打开一个磁盘文件，每次读取其中的 128 个字节块，并将其以十六进制和 ASCII 格式显示到屏幕上。
10. 编写一个函数，它以指定的模式打开一个新的临时文件。在程序结束时，将自动关闭并删除该函数创建的所有临时文件（提示：使用库函数 atexit()）。

www.plcworld.cn

TYPE & RUN 5　计算字符数

该程序名为 count_ch，它打开指定的文本文件，并计算每个字符在其中出现的次数。对于所有的标准键盘字符都进行计算，包括大小写字母、数字、空格和标点。然后，将结果显示在屏幕上。count_ch 不但有一定的用途，同时演示了一些有趣的编程技术。您可以使用操作系统的重定向运算符（>）将输出发送到一个文件中。例如，命令：

```
COUNT_CH > RESULTS.TXT
```

运行该程序，并将结果发送到文件 results.txt 中，而不是显示到屏幕上。然后，您便可以在文本编辑器中查看该文件，也可以打印它。

程序清单 T&R 5　　count_ch.c：一个计算文件中各种字符数的程序

```
1:  /* Counts the number of occurrences
2:     of each character in a file. */
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:  int file_exists(char *filename);
6:  int main( void )
7:  {
8:      char ch, source[80];
9:      int index;
10:     long count[127];
11:     FILE *fp;
12:
13:     /* Get the source and destination filenames. */
14:     fprintf(stderr, "\nEnter source file name: ");
15:     gets(source);
16:
17:     /* See that the source file exists. */
18:     if (!file_exists(source))
19:     {
20:       fprintf(stderr, "\n%s does not exist.\n", source);
21:     exit(1);
22:     }
23:     /* Open the file. */
24:     if ((fp = fopen(source, "rb")) == NULL)
25:     {
26:       fprintf(stderr, "\nError opening %s.\n", source);
27:      exit(1);
28:     }
29:     /* Zero the array elements. */
30:     for (index = 31; index < 127 ; index++)
```

www.plcworld.cn


```
31:       count[index] = 0;
32:
33:     while ( 1 )
34:     {
35:       ch = fgetc(fp);
36:       /* Done if end of file */
37:       if (feof(fp))
38:          break;
39:       /* Count only characters between 32 and 126. */
40:      if (ch > 31 && ch < 127)
41:          count[ch]++;
42:      }
43:     /* Display the results. */
44:     printf("\nChar\t\tCount\n");
45:     for (index = 32; index < 127 ; index++)
46:        printf("[%c]\t%d\n", index, count[index]);
47:     /* Close the file and exit. */
48:     fclose(fp);
49:     return(0);
50: }
51: int file_exists(char *filename)
52: {
53: /* Returns TRUE if filename exists,
54:    FALSE if not. */
55:    FILE *fp;
56:    if ((fp = fopen(filename, "r")) == NULL)
57:       return 0;
58:    else
59:    {
60:       fclose(fp);
61:       return 1;
62:    }
63: }
```

分析：首先来看看第 51～63 行的 file_exists()函数。它接受一个文件名作为参数，如果该文件存在，则返回 TRUE；否则返回 FALSE。该函数中的代码通过以读模式打开该文件来检查它是否存在（第 56 行）。这是一个通用的函数，可将其用于其他程序中。

接下来请注意，为何使用 fprintf()而不是 printf()（如第 14 行）来将消息打印到屏幕上。因为 printf()总是将输出发送到 stdout，如果执行程序时使用了重定向运算符将输出发送到磁盘文件中，则用户将看不到任何消息。使用 fprintf()可以将消息发送到 stderr，而 stderr 总是与屏幕相连。

最后，来看看如何将字符的 ASCII 码用作数组的索引（第 40 和 41 行）。例如，ASCII 码 32 对应的字符为空格，因此将文件中包含的空格总数存储在元素 count[32]中。

运行该程序时，将提示您输入一个文件名：

```
Enter source file name: count_ch.c
```

您输入文件名后，程序将显示各种字符在该文件中出现的次数。下面是输入 count_ch.c 时，该程序的输出：

```
Char    Count
[ ]     434
[!]     1
["]     14
```

www.plcworld.cn


```
[#]     2
[$]     0
[%]     4
[&]     2
[']     0
[(]     29
[)]     29
[*]     24
[+]     6
[,]     11
[-]     0
[.]     13
[/]     20
[0]     4
[1]     11
[2]     7
[3]     4
[4]     0
[5]     0
[6]     1
[7]     4
[8]     1
[9]     0
[:]     1
[;]     27
[<]     5
[=]     10
[>]     3
[?]     0
[@]     0
[A]     1
[B]     0
[C]     5
[D]     2
[E]     6
[F]     3
[G]     1
[H]     0
[I]     2
[J]     0
[K]     0
[L]     7
[M]     0
[N]     2
[O]     1
[P]     0
[Q]     0
[R]     2
[S]     2
[T]     1
[U]     3
[V]     0
```

www.plcworld.cn


```
[W]     0
[X]     0
[Y]     0
[Z]     1
[[]     6
[\]     11
[]]     6
[^]     0
[_]     3
[`]     0
[a]     26
[b]     5
[c]     35
[d]     25
[e]     100
[f]     50
[g]     4
[h]     23
[i]     62
[j]     0
[k]     1
[l]     27
[m]     9
[n]     69
[o]     40
[p]     18
[q]     0
[r]     51
[s]     43
[t]     59
[u]     25
[v]     1
[w]     2
[x]     19
[y]     3
[z]     0
[{]     6
[|]     0
[}]     6
[~]     0
```

www.plcworld.cn

第 17 天课程　操纵字符串

文本数据（C 语言将其存储为字符串）是很多程序的重要组成部分。以前介绍过 C 程序如何存储字符串以及如何输入和输出字符串。C 语言还提供了用于完成其他字符串操纵的函数。今天的课程介绍以下内容：

- 如何确定字符串的长度？
- 如何复制和拼接字符串？
- 用于对字符串进行比较的函数；
- 如何查找字符串？
- 如何转换字符串？
- 如何检测字符？

17.1　确定字符串的长度

前面的课程中介绍过，在 C 程序中，字符串是一个字符序列，其开头用指针标识，末尾用空字符标识。有时候，您需要知道字符串的长度——它包含的字符数。为此，可以使用库函数 strlen()。该函数的原型如下，它位于 string.h 中：

```
size_t strlen(char *str);
```

您可能不知道类型 size_t 是什么。这种类型是在 string.h 中定义的，它是 unsigned 类型，因此函数 strlen() 返回一个无符号整数。很多字符串函数都使用 size_t 类型。您只需记住，它表示 unsigned 类型。

strlen()接受一个指针参数，它指向您要知道其长度的字符串。该函数返回 str 所指的位置和下一个空字符之间的字符数（不包括空字符）。程序清单 17.1 演示了 strlen()函数的用法。

程序清单 17.1　　strlen.c：使用 strlen()函数确定字符串的长度

```
1:  /* Using the strlen() function. */
2:
3:  #include <stdio.h>
4:  #include <string.h>
5:
6:  int main( void )
7:  {
8:     size_t length;
9:     char buf[80];
10:
11:     while (1)
12:     {
13:        puts("\nEnter a line of text, a blank line to exit.");
14:        gets(buf);
15:
```

www.plcworld.cn


```
16:        length = strlen(buf);
17:
18:        if (length != 0)
19:           printf("\nThat line is %u characters long.", length);
20:        else
21:           break;
22:     }
23:     return 0;
24: }
```

该程序的输出如下：

```
Enter a line of text, a blank line to exit.
Just do it!

That line is 11 characters long.
Enter a line of text, a blank line to exit.
```

分析：该程序只是演示 strlen()的用法而已。第 13 和 14 行显示一条消息，并读取一个字符串到 buf 中。第 16 行使用 strlen()来计算 buf 的长度，然后将结果赋给变量 length。第 18 行检查 length 是否为 0，以确定字符串是否为空。如果该字符串不为空，则第 19 行打印其长度。

17.2　复制字符串

C 语言库中包含两个用于复制字符串的函数。由于 C 语言处理字符串的方式，您不能像其他计算机语言中那样将一个字符串赋给另一个，而必须从源字符串所在的内存单元中复制它，然后将它存储到目标字符串所在的内存单元中。字符串复制函数包括 strcpy()和 strncpy()。要使用这些函数，必须包含头文件 string.h。

17.2.1　strcpy()函数

库函数 strcpy()将整个字符串复制到另一个内存单元中，其原型如下：

```
char *strcpy( char *destination, const char *source );
```

函数 strcpy()将 source 指向的字符串（包括末尾的空字符）复制到 destination 指向的位置。返回值为一个指向新的字符串的指针——destination。

使用 strcpy()时，必须首先为目标字符串分配存储空间。该函数无法知道 destination 指向的位置是否是分配好的空间。如果不是，该函数将覆盖从 destination 指向的位置开始的 strlen(source)个字节。这可能导致无法预测的问题。程序清单 17.2 演示了 strcpy()的用法。



注意：使用 malloc()来分配内存时（就像程序清单 17.2 那样），一种良好的编程习惯是，当不再需要使用它时，应使用 free()函数将其释放。free()函数将在第 20 天的课程中介绍。

程序清单 17.2　　strcpy.c：使用 strcpy()之前，必须为目标字符串分配存储空间

```
1:  /* Demonstrates strcpy(). */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:  #include <string.h>
5:
6:  char source[] = "The source string.";
7:
```

www.plcworld.cn


```
8:  int main( void )
9:  {
10:     char dest1[80];
11:     char *dest2, *dest3;
12:
13:     printf("\nsource: %s", source );
14:
15:     /* Copy to dest1 is okay because dest1 points to */
16:     /* 80 bytes of allocated space. */
17:
18:     strcpy(dest1, source);
19:     printf("\ndest1:  %s", dest1);
20:
21:     /* To copy to dest2 you must allocate space. */
22:
23:     dest2 = (char *)malloc(strlen(source) +1);
24:     strcpy(dest2, source);
25:     printf("\ndest2:  %s\n", dest2);
26:
27:     /* Copying without allocating destination space is a no-no. */
28:     /* The following could cause serious problems. */
29:
30:     /* strcpy(dest3, source); */
31:     return 0;
32: }
```

该程序的输出如下：

```
source: The source string.
dest1:  The source string.
dest2:  The source string.
```

该程序演示了如何将字符串复制到字符数组（如第 10 行声明的 dest1）和字符指针（如第 11 行声明的 dest2）。第 13 行打印原来的字符串。然后使用 strcpy()将该字符串复制到 dest1 中（第 18 行）。第 24 行将源字符串复制到 dest2 中。然后，打印 dest1 和 dest2，以表明函数已成功地执行。第 23 行使用 malloc()函数给 dest2 分配适量的内存空间。如果将一个字符串复制给字符指针，而之前又没有给该指针分配适量的内存空间，则结果将是不可预测的。

17.2.2 strncpy()函数

函数 strncpy()类似于 strcpy()，只是让您能够指定要复制多少个字符。该函数的原型如下：

```
char *strncpy(char *destination, const char *source, size_t n);
```

参数 destination 和 source 分别是指向源字符串和目标字符串的指针。该函数最多将 source 中的前 n 个字符串复制到 destination 中。如果 source 中包含的字符少于 n 个，则在后面加上足够数量的空字符，使得复制到 destination 中的总字符数为 n 个。如果 source 包含的字符多于 n 个，则不在 destination 的末尾加上空字符。该函数的返回值为 destination。

程序清单 17.3 演示了 strncpy()的用法。

程序清单 17.3　　strncpy.c：使用 strncpy()函数

```
1:  /* Using the strncpy() function. */
2:
3:  #include <stdio.h>
```

www.plcworld.cn


```
4:  #include <string.h>
5:
6:  char dest[] = ".........................";
7:  char source[] = "abcdefghijklmnopqrstuvwxyz";
8:
9:  int main( void )
10: {
11:     size_t n;
12:
13:     while (1)
14:     {
15:        puts("Enter the number of characters to copy (1-26)");
16:        scanf("%d", &n);
17:
18:        if (n > 0 && n< 27)
19:           break;
20:     }
21:
22:     printf("\nBefore strncpy destination = %s", dest);
23:
24:     strncpy(dest, source, n);
25:
26:     printf("\nAfter strncpy destination = %s\n", dest);
27:     return 0;
28: }
```

该程序的运行情况如下：

```
Enter the number of characters to copy (1-26)
 15

 Before strncpy destination = .........................
 After strncpy destination = abcdefghijklmno...........
```

分析：除了演示 strncpy()的用法外，该程序还演示了一种高效的、确保用户输入正确信息的方式。第 13～20 行的 while 循环提示用户输入一个 1～26 的数字。该循环不断执行，直到用户输入了一个有效的值，因此在用户输入有效的值之前，程序将不会向前运行。当用户输入一个 1～26 的值后，第 22 行打印 dest 的值。然后，第 24 行将从 source 中复制用户指定数目的字符到 dest 中，并打印 dest 的值（第 26 行）。



警告：一定要确保复制的字符数不超过为目标字符串分配的空间。另外，还要注意 strncpy() 不添加空字符的情况。

17.2.3　strdup()函数

有必要介绍一下另一个字符串复制函数。strdup()函数类似于 strcpy()，只是它调用 malloc()来为目标字符串分配内存。实际上，其效果与程序清单 17.2 相同：使用 malloc()分配空间，然后调用 strcpy()。

需要注意的是，strdup()不是 ANSI 标准函数。很多编译器都提供这个函数，包括 Microsoft、Borland 和 Symantec 公司的编译器，但其他 C 编译器可能不提供（或者功能不同）。

strdup()的原型如下：

```
char *strdup( char *source );
```

www.plcworld.cn

其中参数 source 是指向源字符串的指针。该函数返回一个指向目标字符串（malloc()分配的空间）的指针；如果分配内存时失败，则返回 NULL。程序清单 17.4 演示了 strdup()的用法。

程序清单 17.4　　strdup.c：使用 strdup()来复制字符串，并动态分配内存

```
1:  /* The non-ANSI strdup() function. */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:  #include <string.h>
5:
6:  char source[] = "The source string.";
7:
8:  int main( void )
9:  {
10:     char *dest;
11:
12:     if ( (dest = strdup(source)) == NULL)
13:     {
14:        fprintf(stderr, "Error allocating memory.");
15:        exit(1);
16:     }
17:
18:     printf("The destination = %s\n", dest);
19:     return 0;
20: }
```

该程序的输出如下：

```
The destination = The source string.
```

分析：在该程序清单中，strdup()首先为 dest 分配适量的内存，然后复制字符串 source。第 18 行打印复制的字符串。

17.3　拼接字符串

如果您不熟悉术语"拼接"（concatenation），则可能会问：这是什么？合法吗？它指的是将两个字符串连接起来——将一个字符串加在另一个字符串的后面。大多数情况下，这是合法的。C 标准库包含两个字符串拼接函数：strcat()和 strncat()，使用它们时，需要包含头文件 string.h。

17.3.1　strcat()函数

函数 strcat()的原型如下：

```
char *strcat(char *str1, const char *str2);
```

该函数将 str2 加到 str1 的后面，并将空字符移到新字符串的末尾。您必须为 str1 分配足够的空间，以便能够存储拼接后的字符串。strcat()的返回值为 str1。程序清单 17.5 演示了 strcat()函数的用法。

程序清单 17.5　　strcat.c：使用 strcat()拼接字符串

```
1:  /* The strcat() function. */
2:
3:  #include <stdio.h>
4:  #include <string.h>
5:
```

www.plcworld.cn


```
6:  char str1[27] = "a";
7:  char str2[2];
8:
9:  int main( void )
10: {
11:     int n;
12:
13:     /* Put a null character at the end of str2[]. */
14:
15:     str2[1] = '\0';
16:
17:     for (n = 98; n< 123; n++)
18:     {
19:         str2[0] = n;
20:         strcat(str1, str2);
21:         puts(str1);
22:     }
23:     return 0;
24: }
```

该程序的输出如下：

```
ab
abc
abcd
abcde
abcdef
abcdefg
abcdefgh
abcdefghi
abcdefghij
abcdefghijk
abcdefghijkl
abcdefghijklm
abcdefghijklmn
abcdefghijklmno
abcdefghijklmnop
abcdefghijklmnopq
abcdefghijklmnopqr
abcdefghijklmnopqrs
abcdefghijklmnopqrst
abcdefghijklmnopqrstu
abcdefghijklmnopqrstuv
abcdefghijklmnopqrstuvw
abcdefghijklmnopqrstuvwx
abcdefghijklmnopqrstuvwxy
abcdefghijklmnopqrstuvwxyz
```

分析：字母 b～z 的 ASCII 码为 98～122。该程序使用这些 ASCII 码来演示 strcat()的用法。第 17～22 行的 for 循环依次将这些值赋给 str2[0]。由于 str2[1]包含的是空字符（第 15 行），因此相当于依次将字符串“b”、“c”等赋给 str2。这些字符串被依次与 str1 拼接在一起（第 20 行），然后将 str1 显示到屏幕上（第 21 行）。

www.plcworld.cn

17.3.2 strncat()函数

库函数 strncat()也用于拼接字符串，但它让您指定将源字符串中的多少个字符加到目标字符串的后面。该函数的原型如下：

```
char *strncat(char *str1, const char *str2, size_t n);
```

如果 str2 包含的字符多于 n 个，则前 n 个字符被加到 str1 的后面；如果少于 n 个，则 str2 中的所有字符都将被加到 str1 的后面。无论是哪种情况，都将在拼接得到的字符串后面加上空字符。您必须为 str1 分配足够的空间，以便能够存储拼接后的字符串。该函数返回 str1。程序清单 17.6 使用 strncat()来实现程序清单 17.5 的功能。

程序清单 17.6　　strncat.c：使用 strncat()函数拼接字符串

```
1:  /* The strncat() function. */
2:
3:  #include <stdio.h>
4:  #include <string.h>
5:
6:  char str2[] = "abcdefghijklmnopqrstuvwxyz";
7:
8:  int main( void )
9:  {
10:     char str1[27];
11:     int n;
12:
13:     for (n=1; n< 27; n++)
14:     {
15:        strcpy(str1, "");
16:        strncat(str1, str2, n);
17:        puts(str1);
18:     }
19:     return 0;
20: }
```

该程序的输出如下：

```
a
ab
abc
abcd
abcde
abcdef
abcdefg
abcdefgh
abcdefghi
abcdefghij
abcdefghijk
abcdefghijkl
abcdefghijklm
abcdefghijklmn
abcdefghijklmno
abcdefghijklmnop
abcdefghijklmnopq
abcdefghijklmnopqr
```

www.plcworld.cn


```
abcdefghijklmnopqrs
abcdefghijklmnopqrst
abcdefghijklmnopqrstu
abcdefghijklmnopqrstuv
abcdefghijklmnopqrstuvw
abcdefghijklmnopqrstuvwx
abcdefghijklmnopqrstuvwxy
abcdefghijklmnopqrstuvwxyz
```

分析：您可能会问：第 15 行有何用途。这行代码将一个空字符串（只包含一个空字符）复制到 str1 中。这样，str1 中的第一个字符（str1[0]）为空字符。也可以使用语句 str1[0] = 0;或 str1[0] = '\0';来完成这样的任务。

17.4　比较字符串

可以对字符串进行比较，看它们是否相等。如果不相等，则其中的一个字符串将“大于”或“小于”另一个。判断“大于”还是“小于”是基于字符的 ASCII 码进行的。对于字母，这对应于字母顺序；不过所有的大写字符都“小于”小写字母。这是因为大写字母 A～Z 的 ASCII 码为 65～90；而小写字母 a～z 的 ASCII 码为 97～122。因此，对于这些 C 函数来说，“ZEBRA”将“小于”“apple”。

ANSI C 库提供了两种比较字符串的函数：一种比较整个字符串；另一种比较字符串的前几个字符。

17.4.1　比较两个完整字符串

函数 strcmp()逐字符地对两个字符串进行比较，其原型如下：

```
int strcmp(const char *str1, const char *str2);
```

其中参数 str1 和 str2 是指针，分别指向要比较的字符串。该函数的返回值如表 17.1 所示。您可能注意到了，这两个字符串都被作为常量传递给函数，因为不需要修改它们。程序清单 17.7 演示了 strcmp()的用法。

表 17.1　　strcmp()的返回值

返 回 值	含　义
<0	str1 小于 str2
0	str1 等于 str2
>0	str1 大于 str2

程序清单 17.7　　strcmp.c：使用 strcmp()比较字符串

```
1:  /* The strcmp() function. */
2:
3:  #include <stdio.h>
4:  #include <string.h>
5:
6:  int main( void )
7:  {
8:      char str1[80], str2[80];
9:      int x;
10:
11:     while (1)
12:     {
13:
14:         /* Input two strings. */
15:
```

www.plcworld.cn


```
16:       printf("\n\nInput the first string, a blank to exit: ");
17:       gets(str1);
18:
19:       if ( strlen(str1) == 0 )
20:          break;
21:
22:       printf("\nInput the second string: ");
23:       gets(str2);
24:
25:       /* Compare them and display the result. */
26:
27:       x = strcmp(str1, str2);
28:
29:       printf("\nstrcmp(%s,%s) returns %d", str1, str2, x);
30:    }
31:    return 0;
32: }
```

该程序的运行情况如下：

```
Input the first string, a blank to exit: First string

Input the second string: Second string

strcmp(First string,Second string) returns -1

Input the first string, a blank to exit: test string

Input the second string: test string

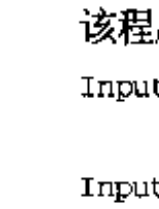
strcmp(test string,test string) returns 0

Input the first string, a blank to exit: zebra

Input the second string: aardvark

strcmp(zebra,aardvark) returns 1

Input the first string, a blank to exit:
```



> 注意：在有些UNIX系统中，当被比较的字符串不同时，字符串比较函数不一定返回-1，但肯定会返回一个非零值。
> ANSI标准只是指出，返回值将小于、等于或大于0。

分析：该程序演示了strcmp()的用法。它提示用户输入两个字符串（第16～17和第22～23行），并将strcmp()返回的结果显示到屏幕上（第29行）。请运行该程序，以便对strcmp()如何比较字符串有一定的感性认识。请输入两个除大小写外相同的字符串，如Smith和SMITH。您知道，strcmp()是区分大小写的，这意味着该程序将大写和小写视为不同。

17.4.2 比较字符串的一部分

库函数strncmp()用于比较两个字符串中的前几个字符，其原型如下：

www.plcworld.cn


```
int strncmp(const char *str1, const char *str2, size_t n);
```

函数 strncmp()对字符串 str1 和 str2 的前 n 个字符进行比较。比较将不断进行下去，直到比较了 n 个字符或到达字符串的末尾。比较的方式和返回值与 strcmp()相同。比较时区分大小写。程序清单 17.8 演示了 strncmp()的用法。

程序清单 17.8　　strncmp.c：使用 strncmp() 比较字符串的一部分

```
1:  /* The strncmp() function. */
2:
3:  #include <stdio.h>
4:  #include <string.h>
5:
6:  char str1[] = "The first string.";
7:  char str2[] = "The second string.";
8:
9:  int main( void )
10: {
11:    size_t n, x;
12:
13:    puts(str1);
14:    puts(str2);
15:
16:    while (1)
17:    {
18:       puts("\n\nEnter number of characters to compare, 0 to exit.");
19:       scanf("%d", &n);
20:
21:       if (n <= 0)
22:          break;
23:
24:       x = strncmp(str1, str2, n);
25:
26:       printf("\nComparing %d characters, strncmp() returns %d.", n, x);
27:    }
28:    return 0;
29: }
```

该程序的运行情况如下：

```
The first string.
The second string.

Enter number of characters to compare, 0 to exit.
3

Comparing 3 characters, strncmp() returns 0.

Enter number of characters to compare, 0 to exit.
6

Comparing 6 characters, strncmp() returns -1.

Enter number of characters to compare, 0 to exit.
```

www.plcworld.cn

0

分析：该程序对第 6 和 7 行定义的两个字符串进行比较。第 13 和 14 行将这两个字符串打印到屏幕上，以便用户知道它们的内容。然后，程序执行一个 while 循环（第 16～27 行），以便用户能进行多次比较。如果用户要求比较 0 个字符（第 18～19 行），则跳出循环（第 22 行），从而结束程序；否则进行比较（第 24 行），并将结果打印到屏幕上（第 26 行）。

17.4.3 比较字符串时忽略大小写

不幸的是，ANSI C 库没有提供任何不区分大小写的字符串比较函数；幸运的是，大多数 C 编译器都提供了用于完成这种任务的函数。Symantec 使用函数 strcmpl()；Microsoft 提供了函数_stricmp()；而 Borland 提供了两个这样的函数：strcmpi()和 stricmp()。要了解您的编译器提供的这种函数，请参阅库参考手册。使用不区分大小写的字符串比较函数时，字符串 Smith 和 SMIT 将相等。请修改程序清单 17.7 中的第 27 行，使用您的编译器提供的不区分大小写的字符串比较函数，然后再次运行该程序。

17.5 查找字符串

C 语言库包含了大量用于查找字符串的函数。换句话说，这些函数检查一个字符串是否出现在另一个字符串中，如果是，在什么位置。可供选择的字符串查找函数有 6 个，要使用其中的任何一个，都必须包含头文件 string.h：

- strchr()；
- strrchr()；
- strcspn()；
- strspn()；
- strpbrk()；
- strstr()。

17.5.1 strchr()函数

strchr()函数找到字符在字符串中第一次出现的位置，其原型如下：

```
char *strchr(constchar *str, int ch);
```

函数 strchr()在 str 中从左到右进行查找，直到找到字符 ch 或者到达结尾的空字符。如果找到 ch，则返回一个指向该字符的指针；否则返回 NULL。

strchr()找到指定的字符时，将返回一个指向该字符的指针。由于 str 是指向字符串中第一个字符的指针，因此可以将 strchr()返回的指针与 str 相减，来获悉该字符所在的位置。程序清单 17.9 说明了这一点。别忘了，字符串中第一个字符的位置为 0。与很多其他的字符串函数一样，strchr()也是区分大小写的，因此将指出，字符串“raffle"中没有“F”。

程序清单 17.9　　strchr.c：使用 strchr()在字符串中查找一个字符

```
1:  /* Searching for a single character with strchr(). */
2:
3:  #include <stdio.h>
4:  #include <string.h>
5:
6:  int main( void )
7:  {
8:      char *loc, buf[80];
9:      int ch;
```

www.plcworld.cn


```
10:
11:    /* Input the string and the character. */
12:
13:    printf("Enter the string to be searched: ");
14:    gets(buf);
15:    printf("Enter the character to search for: ");
16:    ch = getchar();
17:
18:    /* Perform the search. */
19:
20:    loc = strchr(buf, ch);
21:
22:    if ( loc == NULL )
23:       printf('The character %c was not found.", ch);
24:    else
25:       printf("The character %c was found at position %d.\n",
26:              ch, loc-buf);
27:    return 0;
28: }
```

该程序的运行情况如下：

```
Enter the string to be searched: How now Brown Cow?
Enter the character to search for: C
The character C was found at position 14.
```

分析：该程序的第 20 行使用 strchr()在字符串中查找一个字符。strchr()返回一个指针，该指针指向该字符第一次出现的位置；如果没有找到这样的字符，则返回 NULL。第 22 行检查 loc 的值是否为 NULL，并根据情况打印一条相应的消息。正如前面指出的，可以将 strchr()函数的返回值与字符串指针相减来确定该字符在字符串的什么位置。

17.5.2　strrchr()函数

库函数 strrchr()与 strchr()相同，只是它查找指定的字符在字符串中最后一次出现的位置。该函数的原型如下：

```
char *strrchr(const char *str, int ch);
```

函数返回一个指针，该指针指向字符 ch 在字符串 str 中最后一次出现的位置；如果没有找到 ch 字符，则返回 NULL。要查看该函数的运行情况，请修改程序清单 17.9 中的第 20 行，用 strrchr()代替 strchr()。

17.5.3　strcspn()函数

库函数 strcspn()在一个字符串中查找另一个字符串中的字符第一次出现的位置，其原型如下：

```
size_t strcspn(const char *str1, const char *str2);
```

函数 strcspn()从 str1 的第一个字符开始搜索，查找出现在 str2 中的字符。记住这一点很重要。该函数并不查找字符串 str2，而是查找 str2 中的字符。如果找到这样的字符，则返回该字符离 str1 开头的距离；如果没有找到，则返回 strlen(str1)。这表明第一个匹配的字符是该字符串末尾的空字符。程序清单 17.10 演示了如何使用 strcspn()函数。

程序清单 17.10　　strcspn.c：使用 strcspn()查找一组字符

```
1:  /* Searching with strcspn(). */
2:
3:  #include <stdio.h>
4:  #include <string.h>
```

www.plcworld.cn


```
5:
6:  int main( void )
7:  {
8:      char  buf1[80], buf2[80];
9:      size_t loc;
10:
11:     /* Input the strings. */
12:
13:     printf("Enter the string to be searched: ");
14:     gets(buf1);
15:     printf("Enter the string containing target characters: ");
16:     gets(buf2);
17:
18:     /* Perform the search. */
19:
20:     loc = strcspn(buf1, buf2);
21:
22:     if ( loc ==  strlen(buf1) )
23:        printf("No match was found.");
24:     else
25:        printf("The first match was found at position %d.\n", loc);
26:     return0;
27: }
```

该程序的运行情况如下：

```
Enter the string to be searched: How now Brown Cow?
Enter the string containing target characters:  Cat
The first match was found at position 14.
```

分析：该程序清单与程序清单17.9类似，但不是查找一个字符第一次出现的位置，而是查找另一个字符串中的任何字符第一个出现的位置。该程序的第20行调用strcspn()，并将buf1和buf2作为参数传递给它。如果buf2中的任何字符出现在了buf1中，则strcspn()将返回第一次出现的位置离buf1开头的距离。第22行检查返回的值是否为NULL。如果是，则说明没有找到匹配的字符，因此第23行显示一条相应的消息；如果不是，则也显示一条显示，指出该字符在字符串中的什么位置。

17.5.4 strspn()函数

该函数与strcspn()类似，其原型如下：

```
size_t strspn(const char *str1, const char *str2);
```

函数strspn()在str1中进行查找，将str1中的每个字符同str2中的字符进行比较。它返回str1中第一个不被包含在str2中的字符的位置。换句话说，strspn()指出，从str1的开头到什么位置之间的所有字符都包含在str2中。如果str1的第一个字符就不包含在str2中，则该函数返回0。程序清单17.11演示了strspn()的用法。

程序清单17.11　　strspn.c：使用strspn()查找第一个不匹配的字符

```
1:  /* Searching with strspn(). */
2:
3:  #include <stdio.h>
4:  #include <string.h>
5:
6:  int main( void )
7:  {
8:      char  buf1[80], buf2[80];
```

www.plcworld.cn


```
 9:    size_t loc;
10:
11:    /* Input the strings. */
12:
13:    printf("Enter the string to be searched: ");
14:    gets(buf1);
15:    printf("Enter the string containing target characters: ");
16:    gets(buf2);
17:
18:    /* Perform the search. */
19:
20:    loc = strspn(buf1, buf2);
21:
22:    if ( loc == 0 )
23:       printf("No match was found.\n");
24:    else
25:       printf("Characters match up to position %d.\n", loc-1);
26:    return 0;
27: }
```

该程序的运行情况如下：

```
Enter the string to be searched: How now Brown Cow?
Enter the string containing target characters: How now what?
Characters match up to position 7.
```

分析：该程序类似于前一个范例，只不过它在第 20 行调用 strspn()，而不是 strcspn()。该函数返回 buf1 中第一个不包含在 buf2 中的字符离 buf1 开头的距离。第 22～25 行根据返回的值打印相应的消息。

17.5.5　strpbrk()函数

库函数 strpbrk()与 strcspn()类似，它在一个字符串中查找第一个出现在另一个字符串中的字符；不同的是，查找时不考虑结尾的空字符。该函数的原型如下：

```
char *strpbrk( const char *str1, const char *str2);
```

函数 strpbrk()返回一个指针，该指针指向 str1 中第一个出现在 str2 中的字符。如果没有找到这样的字符，则返回 NULL。正如前面介绍 strchr()时指出的，可以将 strpbrk()返回的指针与 str1 相减来获知第一个匹配的字符离 str1 开头的距离。例如，可以将程序清单 17.9 中第 20 行的 strchr()替换为 strpbrk()。

17.5.6　strstr()函数

最后一个（也可能是最有用的）字符串查找函数是 strstr()。该函数查找一个字符串在另一个字符串中第一次出现的位置，它查找的是整个字符串，而不是字符串中的字符。该函数的原型如下：

```
char *strstr(const char *str1, const char *str2);
```

函数 strstr()返回一个指针，该指针指向 str2 在 str1 中第一次出现的位置；如果没有找到，则返回 NULL。如果 str2 的长度为 0，则该函数返回 str1。当函数找到匹配的子字符串时，可以像关于 strchr()的一节中介绍的那样，通过执行指针减法运算来获得它离 str1 开头的距离。查找时，strstr()是区分大小写的。程序清单 17.12 演示了如何使用 strstr()。

程序清单 17.12　　strstr.c：使用 strstr()在一个字符串中查找另一个字符串

```
1:  /* Searching with strstr(). */
2:
3:  #include <stdio.h>
4:  #include <string.h>
```

www.plcworld.cn


```
 5:
 6: int main( void )
 7: {
 8:     char *loc, buf1[80], buf2[80];
 9:
10:     /* Input the strings. */
11:
12:     printf("Enter the string to be searched: ");
13:     gets(buf1);
14:     printf("Enter the target string: ");
15:     gets(buf2);
16:
17:     /* Perform the search. */
18:
19:     loc = strstr(buf1, buf2);
20:
21:     if ( loc == NULL )
22:         printf("No match was found.\n");
23:     else
24:         printf("%s was found at position %d.\n", buf2, loc-buf1);
25:     return 0;
26: }
```

该程序的运行情况如下：

```
Enter the string to be searched: How now brown cow?
Enter the target string: cow
Cow was found at position 14.
```

分析：该函数提供了另一种查找字符串的方式，可以在一个字符串中查找另一个字符串。第12～15行提示用户输入两个字符串。第19行使用strstr()在第一个字符串buf1中查找第二个字符串buf2。如果找到，则返回一个指向buf2第一次出现的位置的指针；否则，返回NULL。第21～24行根据返回的值（loc）打印一条相应的消息。

应　该	不应该
对于这里介绍的很多函数，存在相应的这样的函数，即只在字符串中的前n（由参数指定）个字符中进行查找。这些函数的名称通常为strnxxx()，其中xxx随函数而异。	别忘了，C语言是区分大小写的，因此A和a是不同的。

17.6　字符串转换

很多C语言库提供了两个可用于改变字符串中字符的大小写的函数。这些函数不属于ANSI标准的一部分，因此在您的编译器中，这些函数的名称可能不同，甚至根本没有这样的函数。由于它们很有用，因此这里对其进行介绍。在Microsoft的C编译器中，这些函数的原型如下，它们位于头文件string.h中（在其他编译器中，应该与此类似）：

```
char *strlwr(char *str);
char *strupr(char *str);
```

函数strlwr()将str中的所有大写字母转换为小写；而strupr()则相反，将str中的所有小写字母转换为大写。非字母字符不受影响。这两个函数都返回str。这两个函数都不创建新的字符串，而是对已有的字符串进行修改。程序清单17.13演示了这两个函数的用法。别忘了，编译使用了非ANSI函数的程序时，可能需要

www.plcworld.cn

命令编译器不要执行 ANSI 标准。

程序清单 17.13　　upper.c：使用 strlwr()和 strupr()改变字符串中字符的大小写

```
1:  /* The character conversion functions strlwr() and strupr(). */
2:
3:  #include <stdio.h>
4:  #include <string.h>
5:
6:  int main( void )
7:  {
8:      char buf[80];
9:
10:     while (1)
11:     {
12:         puts("Enter a line of text, a blank to exit.");
13:         gets(buf);
14:
15:         if ( strlen(buf) == 0 )
16:             break;
17:
18:         puts(strlwr(buf));
19:         puts(strupr(buf));
20:     }
21:     return 0;
22: }
```

该程序的运行情况如下：

```
Enter a line of text, a blank to exit.
Bradley L. Jones

bradley l. jones
BRADLEY L. JONES
Enter a line of text, a blank to exit.
```

分析：该程序清单的第 12 行提示用户输入一个字符串，然后对其进行检查，确保它不为空（第 15 行）。第 18 行将字符串转换为小写，然后打印它；第 19 行将字符串转换为大写，然后打印它。

Dev-C++、Symantec、Microsoft 和 Borland 的 C 编译器都提供了这些函数。使用这些函数之前，您应查阅 Library Reference，看您的编译器是否提供了它们。如果要求程序具有可移植性，则不应使用非 ANSI 函数。

17.7　其他字符串函数

本节介绍一些不属于前述各类的字符串函数。要使用这些函数，必须包含头文件 string.h。这些函数是：

- strrev();
- strset();
- strnset()。

17.7.1　strrev()函数

strrev()函数将字符串中所有字符的排列顺序反转，其原型如下：

```
char *strrev(char *str);
```

www.plcworld.cn

除了末尾的空字符不变外，str 中所有字符的排列顺序都将被反转。ANSI 标准并没有定义 strrev()函数，这意味着并非所有的编译器都支持它或者支持的方式可能不同。

该函数返回 str，在下一节介绍 strset()和 strnset()后，程序清单 17.14 演示了 strrev()的用法。

17.7.2 strset()和 strnset()函数

与前一个函数一样，ANSI C 标准库也没有提供 strset()和 strnset()。这些函数将字符串中的所有字符（strset()）或指定数目的字符（strnset()）修改为指定的字符，它们的原型如下：

```
char *strset(char *str, int ch);
char *strnset(char *str, int ch, size_t n);
```

函数 strset()将 str 中除末尾的空字符之外的所有字符修改为 ch；函数 strnset()将 str 中的前 n 个字符修改为 ch。如果 n 大于或等于 strlen(str)，则 strnset()将 str 中的所有字符修改为 ch。程序清单 17.14 演示了这两个函数的用法。

程序清单 17.14　　string.c：演示函数 strrev()、strset()和 strnset()的用法

```
1:  /* Demonstrates strrev(), strset(), and strnset(). */
2:  #include <stdio.h>
3:  #include <string.h>
4:
5:  char str[] = "This is the test string.";
6:
7:  int main( void )
8:  {
9:     printf("\nThe original string: %s", str);
10:    printf("\nCalling strrev(): %s", strrev(str));
11:    printf("\nCalling strrev() again: %s", strrev(str));
12:    printf("\nCalling strnset(): %s", strnset(str, '!', 5));
13:    printf("\nCalling strset(): %s", strset(str, '!'));

14:    return 0;
15: }
```

该程序的输出如下：

```
The original string: This is the test string.
Calling strrev(): .gnirts tset eht si sihT
Calling strrev() again: This is the test string.
Calling strnset(): !!!!!is the test string.
Calling strset(): !!!!!!!!!!!!!!!!!!!!!!
```

分析：该程序演示了三个函数的用法，这是通过打印字符串 str 的值进行的。第 9 行打印最初的字符串；第 10 行打印使用 strrev()反转之后的字符串；第 11 行将字符串反转回原来的状态；第 12 行使用 strnset()函数将 str 的前 5 个字符改为惊叹号；最后，第 13 行将字符串中的所有字符改为惊叹号。

虽然这些函数不是 ANSI 标准的一部分，但 Dev-C++、Symantec、Microsoft 和 Borland 的 C 编译器的函数库都包含它们。您应查阅您的编译器的库参考手册，以了解它是否支持这些函数。

17.8 将字符串转换为数字

有时候您需要将以字符串方式表示的数字转换为真正的数值变量。例如，字符串"123"可被转换为一个值为 123 的 int 变量。用于完成这种转换的函数有 4 个。它们的原型都位于头文件 stdlib.h 中，接下来的几节将

www.plcworld.cn

对其进行介绍。

17.8.1　将字符串转换为 int

库函数 atoi()用于将字符串转换为 int 值，该函数的原型如下：

```
int atoi(const char *ptr);
```

函数 atoi()将 ptr 指向的字符串转换为一个 int 值。除了数字外，该字符串的开头还可以包含空白或符号（+或-）。转换从字符串的开头进行，直到遇到不可转换的字符（如字母和标点符号）为止。转换得到的 int 值将被返回给调用程序。如果字符串中的第一个字符就是不可转换的，则返回 0。表 17.2 是一些例子。

表 17.2　　使用 atoi()将字符串转换为 int 值

字 符 串	atoi()返回的值
"157"	157
"-1.6"	-1
"+50x"	50
"twelve"	0
"x506"	0

第一个例子很简单。对于第二个例子，您可能会问，为何对“.6”进行转换。别忘了，这是将字符串转换为 int 值，数字的小数部分将被丢弃。

第三个例子也很简单，该函数能够识别加号，并将其视为数字的组成部分。第四个例子试图转换“twelve”，atoi()函数不识字，在它眼里只有字符。而该字符串并不以数字打头，因此函数返回 0。对于最后一个例子，情况也是如此。

17.8.2　将字符串转换为 long 值

库函数 atol()的工作原理与 atoi()相同，只是它返回一个 long 值。该函数的原型如下：

```
long atol(const char *ptr);
```

对于表 17.2 的例子，atoll()的返回值与 atoi()相同，只不过返回值的类型为 long，而不是 int。

17.8.3　将字符串转换为 long long 值

与 atoi()和 atol()一样，函数 atoll()将字符串转换为一个 long long 值。该函数的原型如下：

```
long long atoll(const char *ptr);
```

17.8.4　将字符串转换为浮点数

函数 atof()将字符串转换为 double 值，其原型如下：

```
double atof(const char *str);
```

其中参数 str 是一个指针，它指向要转换的字符串。该字符串的开头可以包含空白和符号（+或-）。数字中可以包含 0～9、小数点和指数指示符（E 或 e）。如果第一个字符是不可转换的，则 atof()返回 0。表 17.3 列出了一些 atof()的使用范例。

表 17.3　　使用 atof()将字符串转换为数字

字 符 串	atof()返回的值
"12"	12.000000
"-0.123"	-0.123000
"123E+3"	123000.000000
"123.1e-5"	0.001231

程序清单 17.15 演示了如何使用 atof()。它让用户输入一个字符串，然后对其进行转换。

www.plcworld.cn

程序清单 17.15　　atof.c：使用 atof()将字符串转换为 double 值

```
1:  /* Demonstration of atof(). */
2:
3:  #include <string.h>
4:  #include <stdio.h>
5:  #include <stdlib.h>
6:
7:  int main( void )
8:  {
9:      char buf[80];
10:     double d;
11:
12:     while (1)
13:     {
14:         printf("\nEnter the string to convert (blank to exit):     ");
15:         gets(buf);
16:
17:         if ( strlen(buf) == 0 )
18:             break;
19:
20:         d = atof( buf );
21:
22:         printf("The converted value is %f.", d);
23:     }
24:     return 0;
25: }
```

该程序的运行情况如下：

```
Enter the string to convert (blank to exit):     1009.12
The converted value is 1009.120000.
Enter the string to convert (blank to exit):     abc
The converted value is 0.000000.
Enter the string to convert (blank to exit):     3
The converted value is 3.000000.
Enter the string to convert (blank to exit):
```

分析：第 12～23 行的 while 循环使得程序不断地运行，直到用户输入一个空行。第 14 和 15 行提示用户输入一个字符串。第 17 行检查用户输入的是否为空行。如果是，则跳出 while 循环，然后结束程序。第 20 行调用 atof()，将用户输入的字符串转换为一个 double 值。第 22 行打印转换后的结果。

17.9　字符检测函数

头文件 ctype.h 包含很多用于检测字符的函数的原型，这些函数根据字符是否满足特定的条件，而返回 TRUE 或 FALSE。例如，是否是字母或数字？isxxxx()函数实际上是 ctype.h 中定义的宏。有关宏的概念，将在第 21 天的课程中介绍。那时，您可以查看 ctype.h 中的定义，看看这些函数是如何工作的。就现在而言，您只需知道如何使用它们即可。

所有 isxxxx()宏的原型都相同，都是：

```
int isxxxx(int ch);
```

其中 ch 是要检测的字符。如果条件满足，则返回 TRUE（非零）；否则返回 FALSE（零）。表 17.4 列出

www.plcworld.cn

了所有的 isxxxx()宏。

表 17.4　　isxxxx()宏

宏	功　能
isalnum()	如果 ch 是字母或数字，则返回 TRUE。
isalpha()	如果 ch 是字母，则返回 TRUE。
isblank()	如果 ch 为空，则返回 TRUE。
iscntrl()	如果 ch 是控制字符，则返回 TRUE。
isdigit()	如果 ch 是数字，则返回 TRUE。
isgraph()	如果 ch 是打印字符（空白除外），则返回 TRUE。
islower()	如果 ch 是小写字母，则返回 TRUE。
isprint()	如果 ch 是打印字符（包括空白），则返回 TRUE。
ispunct()	如果 ch 是标点，则返回 TRUE。
isspace()	如果 ch 是空白字符（空格、水平制表符、垂直制表符、换行符、换页符或回车），则返回 TRUE。
isupper()	如果 ch 是大写字母，则返回 TRUE。
isxdigit()	如果 ch 是十六进制的数字（0~9、A~F、a~f），则返回 TRUE。

使用这些字符检测宏，可以完成很多有趣的工作。程序清单 17.16 中的 get_int()就是一个这样的例子，它从 stdin 读取一个 int 值，并将其返回给调用程序。该函数跳过开头的空白，如果第一个非空白字符不是数字，则返回 0。

程序清单 17.16　　getint.c：使用 isxxxx()宏来实现一个读取一个 int 值的函数

```
1:  /* Using character test macros to create an integer */
2:  /* input function. */
3:
4:  #include <stdio.h>
5:  #include <ctype.h>
6:
7:  int get_int(void);
8:
9:  int main( void )
10: {
11:     int x;
12:     x = get_int();
13:
14:     printf("You entered %d.\n", x);
15: }
16:
17: int get_int(void)
18: {
19:     int ch, i, sign = 1;
20:
21:     /* Skip over any leading white space. */
22:
23:     while ( isspace(ch = getchar()) )
24:         ;
25:
26:     /* If the first character is nonnumeric, unget */
27:     /* the character and return 0. */
```

www.plcworld.cn


```
28:
29:     if (ch != '-' && ch != '+'&& !isdigit(ch)&& ch ! =EOF)
30:     {
31:        ungetc(ch, stdin);
32:        return 0;
33:     }
34:
35:     /* If the first character is a minus sign, set */
36:     /* sign accordingly. */
37:
38:     if (ch == '-')
39:        sign = -1;
40:
41:     /* If the first character was a plus or minus sign, */
42:     /* get the next character. */
43:
44:     if (ch == '+' || ch == '-')
45:        ch = getchar();
46:
47:     /* Read characters until a nondigit is input. Assign */
48:     /* values, multiplied by proper power of 10, to i. */
49:
50:     for (i = 0; isdigit(ch); ch = getchar() )
51:        i = 10 * i + (ch - '0');
52:
53:     /* Make result negative if sign is negative. */
54:
55:     i *= sign;
56:
57:     /* If EOF was not encountered, a nondigit character */
58:     /* must have been read in, so unget it. */
59:
60:     if (ch != EOF)
61:        ungetc(ch, stdin);
62:
63:     /* Return the input value. */
64:
65:     return i;
66: }
```

该程序的运行情况如下：

```
-100
You entered -100.
abc3.145
You entered 0.
9 9 9
You entered 9.
2.5
You entered 2.
```

分析：该程序的第31和61行使用了第14天的课程中介绍过的unget()函数。该函数将一个字符还回到指定的流中。程序下次从这个流读取字符时，归还的字符将是第一个被读取的。之所以必须这样做是因为，

www.plcworld.cn

当 get_int()从 stdin 流中读取一个非数字字符后，应将其归还，以免影响程序以后读取 stdin。

该程序的 main()很简单。第 11 行声明了一个名为 x 的 int 变量，然后第 12 行调用 get_int()函数，并将其返回值赋给变量 x，最后，第 14 行将 x 的值显示到屏幕上。程序的余下部分是 get_int()函数。

get_int()不那么简单。为删除开头的空白，第 23 行使用了一个 while 循环。isspace()宏对 getchar()读取的字符 ch 进行检测。如果它是空白，则读取下一个字符。这一过程将不断重复下去，直到读取的字符不是空白为止。第 29 行检测读取的字符是否可用，其含义是：该字符不是负号、正号、数字或 EOF 吗？如果不是，则执行第 31 行，使用 ungetc()将该字符还回到 stdin 中，然后函数结束，返回到 main()；如果是，则继续向下执行。

第 38～45 行处理数字的符号。第 38 行检查字符是否为负号。如果是，则将变量 sign 设置为-1，该变量用于决定该数字是正还是负（第 55 行）。由于默认为正，因此处理负号后便可继续进行。如果用户输入了符号，则程序读取下一个字符（第 44 和 45 行）。

该函数的关键部分是第 50～51 行的 for 循环。它不断地读取字符，直到读取的字符不为数字为止。乍一看，第 51 行可能不好理解。该行读取字符，并将其转换为数字。将数字字符与字符'0'相减，可以将该字符转换为一个数字（别忘了 ASCII 码）。获得数值后，将其与 10 的幂相乘。for 循环将不断进行，直到读取的不是数字为止。然后第 55 行相应的加上符号。

返回之前，函数需要做一些清理工作。如果最后读取的字符不是 EOF，则将其归还到 stdin 流中（以免影响程序的其他部分读取）。这是在第 61 行完成的。最后第 65 行返回调用程序。

应　该	不应该
一定要充分利用可用的字符串函数。	如果打算将应用程序移植到其他平台，则不要使用非 ANSI 函数。 不要将数字和字符混淆。人们很容易将字符“1”和数字 1 视为一回事。

17.9.1　ANSI 对大小写转换的支持

虽然函数 strlwr()和 strupr()能够将字符串转换为大写和小写，但它们不是 ANSI 标准的一部分。然而，ANSI 标准确实定义了两个用于将信息转换为大写和小写的宏。除了 isxxxx()宏外，ANSI 还定义了两个用于改变字符的大小写的宏：toupper()和 tolower()。程序清单 17.17 演示了这两个宏的用法。

程序清单 17.17　　upper2.c：使用 tolower()和 toupper()改变字符串中字符的大小写

```
1:  /* The character conversion functions strlwr() and strupr(). */
2: #include <ctype.h>
3: #include <stdio.h>
4: #include <string.h>
5:
6: int main( void )
7: {
8:    char buf[80];
9:    int  ctr;
10:
11:    while (1)
12:    {
13:       puts("\nEnter a line of text, a blank to exit.");
14:       gets(buf);
15:
16:       if ( strlen(buf) == 0 )
17:          break;
18:
```

www.plcworld.cn


```
19:        for ( ctr = 0; ctr< strlen(buf); ctr++)
20:        {
21:          printf("%c", tolower(buf[ctr]));
22:        }
23:
24:        printf("\n");
25:        for ( ctr = 0; ctr< strlen(buf); ctr++)
26:        {
27:          printf("%c", toupper(buf[ctr]));
28:        }
29:        printf("\n");
30:    }
31:    return 0;
32: }
```

该程序的运行情况如下：

```
Enter a line of text, a blank to exit.
My aun't name is Carolyn C.
my aun't name is carolyn c.
MY AUN'T NAME IS CAROLYN C.Enter a line of text, a blank to exit.
```

分析：该程序的第 13 行提示用户输入一个字符串，第 14 行使用 gets()来读取它。然后，第 16 行检查字符串是否为空。由于 toupper 和 tolower 宏处理的是一个字符，因此这里显示 buf 中的信息的方式不同于程序清单 17.14。第 19 行使用一个 for 循环来遍历字符串中的各个字符，并改变其大小写。



注意：应尽可能使用 toupper()和 tolower()宏，而不是非 ANSI 函数 strupr()和 strlwr()。您可以使用 toupper()和 tolower()宏来创建自己的函数，这样它们将是 ANSI-顺应的。

17.10 总 结

今天的课程介绍了各种操纵字符串的方式。使用 C 标准库函数（和一些编译器特有的非 ANSI 函数），可以对字符串进行复制、拼接、比较和查找。在大多数编程项目中，都必须完成这样的任务。标准库中还提供了用于转换字符串中字符的大小写以及将字符串转换为数字的函数。最后，C 语言还提供了各种字符检测函数（更准确地说，是宏），用于对单个字符进行检测。通过使用这些宏来检测字符，您可以创建自定义的函数。

17.11 问与答

问：如何知道函数是否是 ANSI 兼容的？

答：大多数编译器都有一个库函数参考手册。该手册列出了所有的库函数及其用法。通常，该手册提供了有关函数兼容性的信息。有时候，描述中不但指出了 ANSI 兼容性，还指出了函数是否与 DOS、UNIX、Windows、C++和 OS/2 兼容（大多数编译器只指出与编译器相关的信息）。

问：今天的课程是否介绍了所有的字符串函数？

答：没有。不过，今天介绍的字符串函数几乎能够满足您所有的需要。要了解其他字符串函数，请参阅编译器的库参考手册。

www.plcworld.cn

问：拼接字符串时，strcat()是否忽略末尾的空白？

答：不。strcat()处理空白的方式与其他字符相同。

问：可以将数字转换为字符串吗？

答：可以。您可以编写一个与程序清单 17.16 的函数类似的函数，或者查看库参考手册，看是否有这样的函数。这样的函数有 itoa()、ltoa()、ultoa()、sprint(f)等。

17.12　作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

17.12.1　小测验

1. 何为字符串的长度？如果确定它？
2. 复制字符串之前，必须确保什么？
3. 拼接是什么意思？
4. 比较字符串时，“一个字符串大于另一个”是什么意思？
5. strcmp()和 strncmp()之间有何差别？
6. strcmp()和 strcmpi()之间有何差别？
7. isascii()用于检测什么情况？
8. 对于下面的 var、表 17.4 中的哪个宏返回 TRUE？

```
int var = 1;
```

9. 对于下面的 x，表 17.4 中的哪个宏返回 TRUE？

```
char x = 65;
```

10. 字符检测函数有何用途？

17.12.2　练习

1. 检测函数返回什么值？
2. 如果将下述值分别传递给函数 atoi()，返回值是什么？
 a. "65"
 b. "81.23"
 c. "-34.2"
 d. "ten"
 e. "+12hundred"
 f. "negative100"
3. 如果将下述值分别传递给函数 atof()，返回值是什么？
 a. "65"
 b. "81.23"
 c. "-34.2"
 d. "ten"
 e. "+12hundred"
 f. "1e+3"
4. 排错：下述代码有何错误？

```
char *string1, string2;
string1 = "Hello World";
strcpy( string2, string1);
```

www.plcworld.cn


```
printf( "%s %s", string1, string2 );
```

由于下面的练习有多种解决方案，因此附录 F 没有提供它们的答案。

5. 编写一个程序，它提示用户输入自己的名、姓和中名，然后将姓名存储在一个新的字符串中，存储方式为：名的首字母、句点、空白、中名首字母、句点、空格和姓。例如，如果用户输入 Bradley、Lee 和 Jones，则将其存储为 B. L. Jones。然后将新的姓名打印到屏幕上。

6. 编写一个程序，来验证问题 8 和问题 9 的答案。

7. 函数 strstr()在一个字符串中查找另一个字符串，它区分大小写。编写一个执行相同任务的函数，但不区分大小写。

8. 编写一个函数，确定一个字符串在另一个字符串中出现的次数。

9. 编写一个程序，它在一个文本文件中查找用户指定的字符串，如果找到，则指出位于哪些行中。例如，如果在一个 C 源代码文件中查找“printf()”，则程序应列出包含 printf()的所有行。

10. 程序清单 17.16 包含一个从 stdin 中读取一个 int 值的函数。请编写一个名为 get_float()的函数，它从 stdin 中读取一个 float 值。

www.plcworld.cn

第 18 天课程　有关函数的高级主题

您知道，函数是C语言编程的核心。今天的课程将介绍其他一些使用函数的方式，包括：

- 将指针作为参数传递给函数；
- 将 void 类型的指针作为参数传递给函数；
- 使用接受可变数目参数的函数；
- 从函数返回指针。

其中的一些主题在本书前面已经提到过，但今天的课程将做更详细的介绍。

18.1　将指针传递给函数

默认情况下，参数是按值传递的。按值传递意味着将参数的一个副本传递给函数。这种方式由三步组成：

1. 计算参数表达式的值；
2. 将结果复制到堆栈（stack）中——内存中的一个临时存储空间；
3. 函数从堆栈中取回参数的值。

这里的关键是，变量作为参数被传递给函数时，函数中的代码将无法修改该变量的值。图 18.1 说明了按值传递的情况。其中，参数是一个 int 变量，但这里的原理同样适用于其他类型的变量和更复杂的表达式。

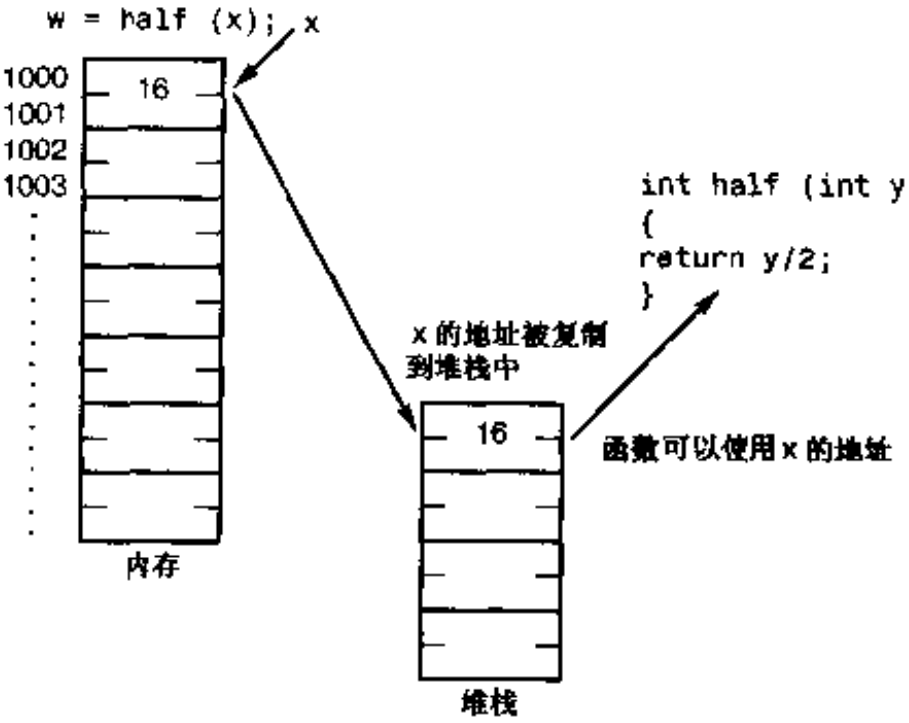


图 18.1　按值传递参数，函数无法修改原来的参数变量

按值将变量传递给函数时，函数可以使用变量的值，但无法访问该变量。因此，函数中的代码无法修改原来的变量。这也就是默认情况下，参数是按值传递的主要原因：防止无意间修改函数外面的数据。

还有另一种传递参数的方式。按值传递适用于基本数据类型（char、short、int、long、long long、float、double 和 long double）和结构。另一种方法是传递指向变量的指针，而不是变量的值。这种传递参数的方法

www.plcworld.cn

被称为按引用传递。由于函数有了变量的地址，因此可以修改它的值。

正如第 9 天的课程中介绍过的，要将数组传递给函数，只能按引用传递；按值传递数组是不可能的。然而，对于其他数据类型，您可以使用任何一种方式进行传递。如果程序使用了大型结构，则按值传递它们可能导致程序耗尽堆栈空间。除这方面外，按引用而不是按值传递参数有利也有弊：

- 按引用传递的优点是，函数可以修改参数变量的值；
- 按引用传递的缺点是，函数可以修改参数变量的值。

您可能会问，优点也是缺点？是的。是优点还是缺点取决于具体情况。如果程序要求函数能够修改参数变量，则按引用传递就是优点；如果没有这种需求，则这样做就是缺点，因为可能无意间修改了参数变量。

您可能会问，为何不使用函数的返回值来修改参数变量呢？当然，您可以这样做，如下面的例子所示：

```
x = half(x);

float half(float y)
{
    return y/2;
}
```

然而，别忘了，函数只能返回一个值。通过按引用传递一个或多个参数，函数能够将多个值“返回”给调用程序。图 18.2 说明了按引用传递一个参数的情况。

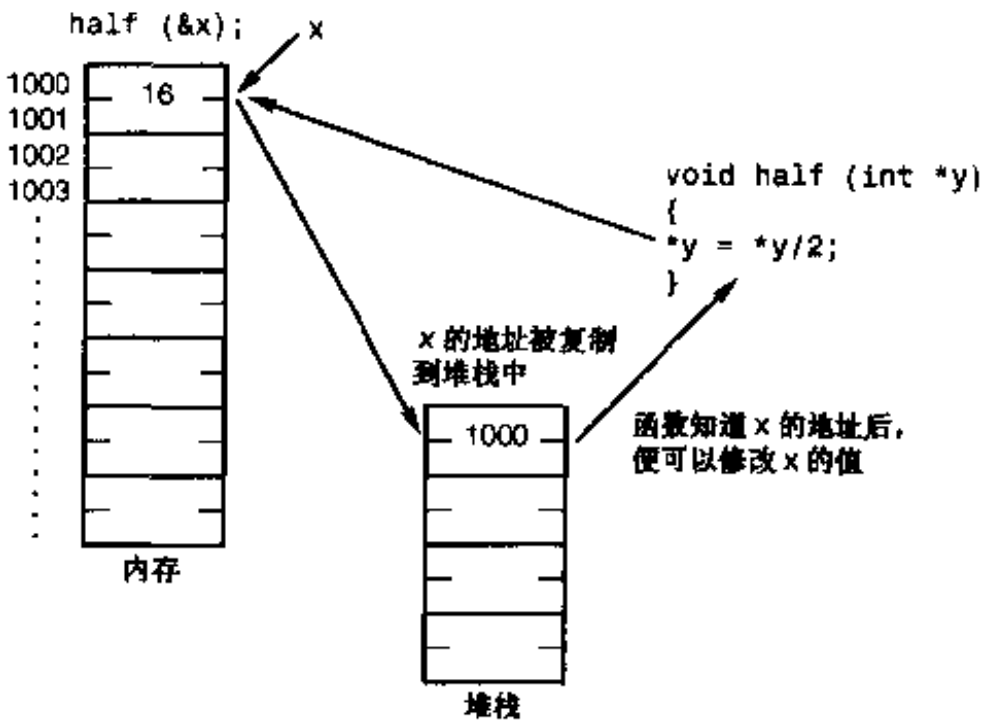


图 18.2 按引用传递使得函数能够修改原来的参数变量

在实际的程序中，对于图 18.2 中的函数并不一定应该按引用传递参数，但它说明了按引用传递的概念。按引用传递时，必须在函数定义和原型中指定该参数是一个指针。在函数体中，必须使用间接运算符来存储按引用传递来的变量。

程序清单 18.1 演示了按引用传递以及默认的按值传递。该程序的输出表明，对于按值传递的变量，函数不能修改它；而对于按引用传递的变量，则可以。当然，函数也可能不需要修改按引用传递来的变量，在这种情况下，就没有理由按引用传递。

程序清单 18.1　　args.c：按值传递和按引用传递

```
1:  /* Passing arguments by value and by reference. */
2:
3:  #include <stdio.h>
4:
5:  void by_value(int a, int b, int c);
```

www.plcworld.cn


```
6:  void by_ref(int *a, int *b, int *c);
7:
8:  int main( void )
9:  {
10:     int x = 2, y = 4, z = 6;
11:
12:     printf("\nBefore calling by_value(), x = %d, y = %d, z = %d.",
13:        x, y, z);
14:
15:     by_value(x, y, z);
16:
17:     printf("\nAfter calling by_value(), x = %d, y = %d, z = %d.",
18:          x, y, z);
19:
20:     by_ref(&x, &y, &z);
21:     printf("\nAfter calling by_ref(), x = %d, y = %d, z = %d.\n",
22:          x, y, z);
23:     return 0;
24: }
25:
26: void by_value(int a, int b, int c)
27: {
28:     a = 0;
29:     b = 0;
30:     c = 0;
31: }
32:
33: void by_ref(int *a, int *b, int *c)
34: {
35:     *a = 0;
36:     *b = 0;
37:     *c = 0;
38: }
```

该程序的输出如下：

```
Before calling by_value(), x = 2, y = 4, z = 6.
After calling by_value(), x = 2, y = 4, z = 6.
After calling by_ref(), x = 0, y = 0, z = 0.
```

该程序演示了按值传递和按引用传递之间的区别。第 5 和 6 行是程序中将调用的两个函数的原型。在第 5 行中的 by_value()函数接受三个 int 参数，而第 6 行的 by_ref()函数接受三个 int 指针参数。这两个函数的函数头位于第 26 和 33 行，它们的格式与原型相同。这两个函数的函数体类似，但不完全相同。两个函数都将 0 赋给传递给它的三个变量。在 by_value()中，直接将 0 赋给这些变量；而在 by_ref()中，使用的是指针，因此赋值前必须对变量解除引用。

每个函数都被 main()调用一次。首先第 10 行将非零值赋给三个要传递的变量。第 12 行将这些值打印到屏幕上。第 15 行调用函数 by_value()。第 17 行再次打印这三个变量的值。注意，它们的值没有变。这三个变量是按值传递给函数 by_value()的，因此函数无法修改它们的值。第 20 行调用 by_ref()，而第 22 行再次打印这三个变量的值。这次，它们的值都变成了 0。按引用传递变量使得 by_ref()能够存取变量。

您可以编写一个这样的函数，即其中的一些参数按引用传递，而其他的按值传递。只是别忘了，在函数中，要存取按引用传递的参数时，必须使用间接运算符（*）。

www.plcworld.cn

应　该	不应该
如果不需要修改变量原来的值，应按值传递它； 对于按引用传递给函数的变量，一定要在函数中使用间接运算符来解除引用。	在不必要的情况下，不要按值传递大量的数据。这样做可能耗尽堆栈空间； 别忘了，按引用传递的参数应该是一个指针。

18.2　void 类型的指针

前面介绍过，在函数声明中，可以使用关键字 void 来指定函数不接受任何参数或不返回任何值。关键字 void 还可用于创建通用（generic）指针——一个可指向任何类型的数据对象的指针。例如，下面的语句：

```
void *ptr;
```

将 ptr 声明为一个通用指针，但没有指定它指向的东西。

void 指针最常见的用途是用于声明函数参数。您可能希望一个函数能够处理不同类型的参数。您可以将 int 变量传递给它，也可以将 float 变量传递给它，等等。如果将该函数声明为接受一个 void 指针作为参数，则它可以接受任何类型的数据：在这种情况下，您可以将指向任何东西的指针传递给该函数。

这里是一个简单的例子：编写一个这样的函数，它接受一个数值变量作为参数，将该变量的值除以 2，然后将结果返回给参数变量。也就是说，如果变量 val 的值为 4，调用 half(val)后，该变量的值将为 2。由于要修改参数，因此按引用传递它。由于您希望函数能够接受任何数值类型的变量，因此您将该函数声明为接受一个 void 指针作为参数：

```
void half(void *val);
```

现在，您便可以调用该函数，并将任何指针作为参数传递给它。然而，还有一项工作没有做。虽然您无需知道 void 指针指向的数据类型，也可以传递它，但您无法对该指针执行解除引用的操作。只有知道指针指向的数据类型后，函数中的代码才能使用它。为此，您可以使用强制类型转换（typecast），强制类型转换只是告诉程序，将该 void 指针视为一个指向特定类型的指针。如果 pval 是一个 void 指针，则可以这样进行强制类型转换：

```
(type *)pval;
```

其中，type 是一种数据类型。要告诉程序：pval 是一个指向 int 类型的指针，可以这样编写代码：

```
(int *)pval;
```

要解除指针的引用（即存取 pval 指向的 int 变量），可以这样编写代码：

```
*(int *)pval
```

有关强制类型转换的更详细的信息，请参阅第 20 天的课程。回到原来的话题（将 void 指针传递给函数），函数要使用指针，必须知道它指向的数据类型。对于您要编写的、将其参数除以 2 的函数而言，可能使用的数据类型有 4 种：int、long、float 和 double。除了传递指向变量（您要将该变量除以 2）的指针外，还必须告诉函数：该指针指向的是哪种数据类型。可以将函数定义修改为如下所示：

```
void half(void *pval, char type);
```

函数将根据参数 type 的值，把 void 指针强制转换为合适的类型。然后便可以解除指针的引用，并使用该指针指向的变量的值。函数 half()的最后版本如程序清单 18.2 所示。

程序清单 18.2　　typecast.c：使用 void 指针将不同的数据类型传递给函数

```
1:  /* Using type void pointers. */
2:
3:  #include <stdio.h>
4:
5:  void half(void *pval, char type);
6:
7:  int main( void )
8:  {
```

www.plcworld.cn


```
    /* Initialize one variable of each type. */

    int i = 20;
    long l = 100000;
    float f = 12.456;
    double d = 123.044444;

    /* Display their initial values. */

    printf("\n%d", i);
    printf("\n%ld", l);
    printf("\n%f", f);
    printf("\n%lf\n\n", d);

    /* Call half() for each variable. */

    half(&i, 'i');
    half(&l, 'l');
    half(&d, 'd');
    half(&f, 'f');

    /* Display their new values. */
    printf("\n%d", i);
    printf("\n%ld", l);
    printf("\n%f", f);
    printf("\n%lf\n", d);
    return 0;
}

void half(void *pval, char type)
{
    /* Depending on the value of type, cast the */
    /* pointer val appropriately and divide by 2. */

    switch (type)
    {
    case 'i':
        {
        *((int *)pval) /= 2;
        break;
        }
    case 'l':
        {
        *((long *)pval) /= 2;
        break;
        }
    case 'f':
        {
        *((float *)pval) /= 2;
        break;
        }
    case 'd':
```

www.plcworld.cn


```
61:            {
62:            *((double *)pval) /= 2;
63:            break;
64:            }
65:    }
66: }
```

该程序的输出如下：

```
20
100000
12.456000
123.044444

10
50000
6.228000
61.522222
```

分析：在该程序清单中，第 38～66 行的 half()函数没有进行错误检查（例如，传递的参数的类型是否合法）。这是因为，在实际的程序中，您不会使用一个函数来执行将一个值除以 2 这样简单的任务。这个范例只是用于演示而已。

您可能认为，将指针指向的变量类型作为参数传递给函数降低了灵活性。如果无需知道指针指向的数据类型，则函数将更通用，但 C 语言并不是以这种方式工作的。要解除 void 指针的引用，必须首先将它转换为特定的类型。通过这种方法，您只需要编写一个函数。如果不使用 void 指针，则需要编写 4 个函数——每种数据类型一个。

当您希望函数能够处理不同的数据类型时，总是可以编写一个宏来代替函数。对于前面这样的例子（函数执行的任务相对简单），就很适合编写一个宏来代替（有关宏，请参见第 21 天的课程）。

应　该	不 应 该
使用 void 指针指向的值之前，一定要转换其类型。	不要对 void 指针执行递增或递减运算。

18.3　接受可变数目参数的函数

您已经使用过多个接受可变数目参数的函数，如 printf()和 scanf()，您也可以编写这样的函数。在程序中使用接受可变数目参数的函数时，必须包含头文件 stdarg.h。

声明接受可变数目参数的函数时，首先列出固定的参数——必不可少的参数（固定参数至少要有一个）。然后，在参数列表的最后加上一个省略号（...），指出可将 0 个或多个其他参数传递给函数。在这里的讨论中，请注意形参和实参之间的区别，第 5 天的课程对此进行了讨论。

函数如何知道传递给它了多少个参数呢？您告诉它。可以使用其中的一个固定参数指出传递了多少个参数。例如，当您使用 printf()函数时，格式化字符串中的转换说明符的数目告诉了函数应期望多少个其他的参数。一种更直接的方法时，使用一个固定参数指出将传递多少个其他的参数。稍后的范例使用的便是这种方法，不过在此之前我们先来看看 C 语言提供的用于处理可变参数列表的工具。

函数还必须知道可变列表中各个参数的类型。在 printf()函数中，转换说明符指出了每个参数的类型。在其他情况下（如后面的范例中），可变列表中的所有参数的类型都相同，因此没有任何问题。要创建数目可变、类型不同的参数的函数，您必须采用某种方法来传递有关参数类型的信息。例如，您可以使用字符代码，就像程序清单 18.2 中的 half()函数那样。

用于使用可变参数列表的工具是在 stdarg.h 中定义的。函数可以使用这些工具取回可变列表中的参数。

www.plcworld.cn

它们是：

- va_list：一种指针数据类型；
- va_start()：一个用于初始化参数列表的宏；
- va_arg()：一个用于依次取回可变列表中每个参数的宏；
- va_end()：一个用于在取回完所有的参数后进行“清理”工作的宏。

简要地介绍这些宏后，接下来提供一个使用这些宏的范例。函数被调用时，其中的代码必须按下面的步骤来存取参数：

1. 声明一个 va_list 类型的指针变量，用于存取各个参数。通常将该变量命名为 arg_ptr，虽然不一定非得这样。

2. 调用 va_start()宏，并将指针 arg_ptr 和最后一个固定参数的名称传递给它。宏 va_start()没有返回值，它初始化指针 arg_ptr，使之指向可变列表中的第一个参数。

3. 要取回每一个参数，可调用 va_arg()，并将指针 arg_ptr 和下一个参数的数据类型传递给它。va_arg()返回下一个参数的值。如果函数从可变列表中收到 n 个参数，则调用 va_arg()n 次，依次取回函数调用中列出的参数。

4. 取回可变列表中的所有参数后，调用 va_end()，并将指针 arg_ptr 传递给它。在有些实现中，这个宏不执行任何操作；但在其他一些实现中，它执行必要的清理工作。您应养成调用 va_end()的习惯，以防万一您使用的 C 实现要求这样做。

现在来看一个例子：程序清单 18.3 中的 average()计算一系列整数的算术平均值。该函数接受一个固定参数，该参数指出后面还有多少个参数。

程序清单 18.3　vary.c：使用可变长度的参数列表

```
1:  /* Functions with a variable argument list. */
2:
3:  #include <stdio.h>
4:  #include <stdarg.h>
5:
6:  float average(int num, ...);
7:
8:  int main( void )
9:  {
10:     float x;
11:
12:     x = average(10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
13:     printf("\nThe first average is %f.", x);
14:     x = average(5, 121, 206, 76, 31, 5);
15:     printf("\nThe second average is %f.\n", x);
16:     return 0;
17: }
18:
19: float average(int num, ...)
20: {
21:     /* Declare a variable of type va_list. */
22:
23:     va_list arg_ptr;
24:     int count, total = 0;
25:
26:     /* Initialize the argument pointer. */
27:
28:     va_start(arg_ptr, num);
```

www.plcworld.cn


```
29:
30:     /* Retrieve each argument in the variable list. */
31:
32:     for (count = 0; count < num; count++)
33:        total += va_arg( arg_ptr, int );
34:
35:     /* Perform clean up. */
36:
37:     va_end(arg_ptr);
38:
39:     /* Divide the total by the number of values to get the */
40:     /* average. Cast the total to type float so the value */
41:     /* returned is type float. */
42:
43:     return ((float)total/num);
44: }
```

该程序的输出如下：

```
The first average is 5.500000.
The second average is 87.800000.
```

分析：函数 average()从第 19 行开始。它的第一个参数（唯一的固定参数），指出可变参数列表中有多少个值。在该函数中，第 32～33 行依次取回可变列表中的每个参数，并将其加入到变量 total 中。取回所有的参数后，第 43 行将 total 的类型强制转换为 float，然后除以 num，从而得到平均值。

有必要指出另外两点。第 28 行调用 va_start()来初始化参数列表。取回参数之前，必须这样做。第 37 行调用 va_end()来完成“清理”工作，因为函数已经使用完这些值。编写接受可变数目的参数的函数时，一定要调用这两个函数。

严格地说，接受可变数目参数的函数并不需要有一个固定参数来告诉它给它传递了多少个参数。例如，您可以使用一个其他地方不使用的特殊值来标记参数列表的末尾。然而，这种方法限制了可传递的参数，最好避免使用。

18.4 返回指针的函数

在前一天的课程中，介绍了 C 标准库中的几个返回值为指针的函数。您也可以编写返回指针的函数。您可能猜到了，声明和定义这种函数时都将使用间接运算符（*）。声明这种函数的通用格式如下：

```
type *func(parameter_list);
```

上述语句声明了一个名为 func()的函数，它返回一个指向 type 类型的指针。下面是两个更具体的例子：

```
double *func1(parameter_list);
struct address *func2(parameter_list);
```

第 1 行声明的函数返回一个 double 指针；第 2 行声明的函数返回一个指向 address（您假设它是用户定义的一种结构）类型的指针。

不要将返回指针的函数与函数指针混淆。如果用一对圆括号将间接运算符和名称括起，则声明的是一个函数指针，如下面的两个范例所示。

```
double (*func)(...);    /* Pointer to a function that returns a double. */
double *func(...);      /* Function that returns a pointer to a double. */
```

了解声明的格式后，如何使用返回指针的函数呢？这种函数没有什么特别——可以像使用其他函数那样使用它们：将它们的返回值赋给一个相应类型（这里为指针）的变量。由于函数调用是一个表达式，因此您可以将其用于任何可使用相应类型的指针的地方。

www.plcworld.cn

程序清单 18.4 是一个简单的例子，一个函数接受两个参数，并比较哪一个大。该程序清单使用了两种方式来完成这项工作：一个函数返回一个 int 值；另一个函数返回一个 int 指针。

程序清单 18.4　　return.c：从函数返回一个指针

```
1:  /* Function that returns a pointer. */
2:
3:  #include <stdio.h>
4:
5:  int larger1(int x, int y);
6:  int *larger2(int *x, int *y);
7:
8:  int main( void )
9:  {
10:     int a, b, bigger1, *bigger2;
11:
12:     printf("Enter two integer values: ");
13:     scanf("%d %d", &a, &b);
14:
15:     bigger1 = larger1(a, b);
16:     printf("\nThe larger value is %d.", bigger1);
17:     bigger2 = larger2(&a, &b);
18:     printf("\nThe larger value is %d.\n", *bigger2);
19:     return 0;
20: }
21:
22: int larger1(int x, int y)
23: {
24:     if (y > x)
25:         return y;
26:     return x;
27: }
28:
29: int *larger2(int *x, int *y)
30: {
31:     if (*y > *x)
32:         return y;
33:
34:     return x;
35: }
```

该程序的运行情况如下：

```
Enter two integer values: 1111 3000

The larger value is 3000.
The larger value is 3000.
```

分析：这个程序比较容易理解。第 5 和 6 行是两个函数的原型。第一个函数（larger1()）接受两个 int 参数，并返回一个 int 值；第二个函数（larger2()）接受两个 int 指针，并返回一个 int 指针。位于第 8～20 行的 main()函数很简单。第 10 行声明了 4 个变量，其中 a 和 b 用于存储两个要比较的值；bigger1 和 bigger2 用于存储函数 larger1()和 larger2()返回的值。注意，bigger2 是一个 int 指针，而 bigger1 是一个 int 变量。

第 15 行调用 larger1()，并将两个 int 变量（a 和 b）传递给它，然后将函数返回的值赋给 bigger1，并在

www.plcworld.cn

第 16 行打印 bigger1 的值。第 17 行调用 larger2()，并将两个 int 变量的地址传递给它。larger2()返回的值（一个指针）被赋给 bigger2（也是一个指针）。接下来的一行对这个值执行解除引用操作，并打印它。

这两个比较函数非常类似。它们都对两个值进行比较，并返回较大的一个；不同之处在于，larger2()使用的是指针，而 larger1()不是。在 larger2()中，比较时使用了解除引用运算符，但第 32 和 34 行的 return 语句没有使用。

在很多情况下（如程序清单 18.4 中），编写返回值和返回指针的函数都是可行的，到底采用哪种方法取决于程序的具体情况——主要是您想如何使用返回值。

应　该	不 应 该
编写接受可变数目参数的函数时，一定要使用今天介绍的所有工具。即使您的编译器不要求，也应该这样做。这些工具是 va_list()、va_start()、va_arg()和 va_end()。	不要将函数指针和返回指针的函数混淆。

18.5　总　结

今天的课程介绍了一些有关函数的高级主题。您知道了按值传递参数和按引用传递参数之间的区别，以及后一种技术如何让函数能够将多个值“返回”给调用程序。您还知道了如何使用 void 类型来创建可以指向任何类型的数据对象的指针。void 指针最常见的用途是用于将不同类型的参数传递给函数。别忘了，对 void 指针解除引用之前，必须将其转换为特定的类型。

今天的课程还介绍了如何使用 stdarg.h 中定义的宏来编写接受可变数目参数的函数，这种函数提供了极大的编程灵活性。最后介绍了如何编写返回指针的函数。

18.6　问与答

问：在 C 语言编程中，经常将指针作为参数传递给函数吗？

答：是的。在很多情况下，函数需要修改多个变量的值，而完成这种任务的方式有两种。第一种方法是声明并使用全局变量；第二种方法是将指针传递给函数，让函数能够直接修改数据。只有程序中的几乎所有函数都需要使用的变量，才应将其声明为全局变量；否则不应这样做（参见第 12 天的课程）。

问：修改变量的值时，下面哪种方式更好：将函数的返回值赋给它；将指向它的指针传递给函数。

答：只需修改一个变量时，前一种方法更好。原因很简单。不传递指针可以避免无意间修改数据的情况发生，并使函数独立于其他代码。

18.7　作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

18.7.1　小测验

1. 将参数传递给函数时，按值传递和按引用传递之间有何区别？
2. void 指针是什么？
3. 指出使用 void 指针的原因之一。
4. 使用 void 指针时，强制类型转换有何作用？何时必须进行强制类型转换？
5. 可以编写只接受一个可变参数列表，而没有固定参数的函数吗？

www.plcworld.cn

6. 编写接受可变数目参数的函数时，需要使用哪些宏？
7. 对 void 指针执行递增运算时，其值将增加多少？
8. 函数可以返回指针吗？
9. 要从传递给函数的可变参数列表中取回值，应使用哪个宏？

18.7.2　练习

1. 为这样的函数编写原型，它接受一个指向字符数组的指针作为参数，并返回一个 int 值。
2. 为函数 numbers 编写原型，它接受三个 int 参数，这些参数按引用传递。
3. 编写这样的代码，即调用练习 2 中的 numbers 函数，并将整数 int1、int2 和 int3 传递给它。
4. 排错：下面的代码有错吗？

```
void squared(void *nbr)
{
    *nbr *= *nbr;
}
```

5. 排错：下面的代码有错吗？

```
float total( int num, ...)
{
    int count, total = 0;
    for ( count = 0; count < num; count++ )
        total += va_arg( arg_ptr, int );
    return ( total );
}
```

下面的练习有多种解决方案，因此附录 F 没有提供它们的答案。

6. 编写一个这样的函数，即将可变数目的字符串作为参数、依次将这些字符串拼接成一个更长的字符串、并返回一个指向拼接成的字符串的指针。

7. 编写一个这样的函数，即可接受任何数值类型的数组作为参数、找到该数组中最大和最小值、并返回指向这些值的指针（提示：需要采用某种方法将数组包含的元素数目告知函数）。

8. 编写一个这样的函数，它将一个字符串和一个字符作为参数，查找字符在字符串中第一次出现的位置，并返回一个指向这个位置的指针。

www.plcworld.cn

第 19 天课程　函数库

正如您在本书中看到的，C 语言的强大功能在很大程度上依赖于标准库。今天的课程将介绍一些与其他课程的主题不相关的函数，包括：

- 数学函数；
- 处理时间的函数；
- 处理错误的函数；
- 查找和排序函数。

19.1　数学函数

C 语言标准库中包含各种用于执行数学运算的函数，这些函数的原型位于头文件 math.h 中。数学函数的返回类型都为 double。在三角函数中，角度的单位为弧度，而不是您常用的单位度。1 弧度等于 57.296 度，1 周（360 度）为 2π 弧度。

19.1.1　三角函数

三角函数用于完成一些图形和工程应用程序中的计算，表 19.1 列出了这些函数。

表 19.1　　三角函数

函　数	原　型	描　述
acos()	double acos(double x)	返回参数的反余弦。参数的取值范围为[-1, 1]，返回值的取值范围为[0, π]。
asin()	double asin(double x)	返回参数的反正弦。参数的取值范围为[-1, 1]，返回值的取值范围为[-π/2, π/2]。
atan()	double atan(double x)	返回参数的反正切。返回值的取值范围为[-π/2, π/2]。
atan2()	double atan2(double x, double y)	返回 x/y 的反正切。返回值的取值范围为[-π, π]。
cos()	double cos(double x)	返回参数的余弦。
sin()	double sin(double x)	返回参数的正弦。
tan()	double tan(double x)	返回参数的正切。

19.1.2　指数函数和对数函数

指数函数和对数函数用于完成一些数学计算，表 19.2 列出了这些函数。

表 19.2　　指数函数和对数函数

函　数	原　型	描　述
exp()	double exp(double x)	返回 e^x，其中 e=2.71828182845904523536。
log()	double log(double x)	返回自然对数，参数必须大于 0。
log10()	double log10(double x)	返回以 10 为底的对数，参数必须大于 0。
frexp()	double frexp(double x, int *y)	该函数返回一个范围为[0.5, 1.0]归一化小数，并将一个整数值赋给 y，使得 x = r * 2^y。如果 x 的值为 0，则 r 和 y 也将为 0。
ldexp()	double ldexp(double x, int y)	返回 x * 2^y。

www.plcworld.cn

19.1.3　双曲线函数

双曲线函数执行双曲三角运算，表 19.3 列出了这些函数。

表 19.3　　双曲函数

函　数	原　型	描　述
cosh()	double cosh(double x)	返回双曲余弦
sinh()	double sinh(double x)	返回双曲正弦
tanh()	double tanh(double x)	返回双曲正切

19.1.4　其他数学函数

标准 C 库还包含如表 19.4 所示的数学函数。

表 19.4　　其他数学函数

函　数	原　型	描　述
sqrt()	double sqrt(double x)	返回平方根，参数必须大于或等于 0
ceil()	double ceil(double x)	返回不小于参数的最小整数。例如，ceil(4.5)返回 5.0，而 ceil(-4.5)返回-4.0。虽然该函数返回一个整数值，但其类型为 double。
abs()	int abs(int x)	返回绝对值。
floor()	double floor(double x)	返回不大于参数的最大整数，例如，floor(4.5)返回 4.0，而 floor(-4.5)返回-5.0。
modf()	double modf(double x, double *y)	将参数分为整数部分和小数部分，每部分的符号与参数相同。函数将小数部分返回，并将整数部分赋给*y。
pow()	double pow(double x, double y)	返回 x^y。如果 x ==0，且 y <=0 或者 x < 0，且 y 不为整数，则将导致错误。
fmod()	double fmod(double x, double y)	返回 x/y 的余数（浮点数），符号与 x 相同；如果 x ==0，则返回 0。

19.1.5　演示数学函数

要演示所有的数学函数，需要一整本书的篇幅。程序清单 19.1 中的程序演示了几个数学函数。

程序清单 19.1　　math.c：使用 C 语言库中的数学函数

```
1:  /* Demonstrates some of C's ,math functions */
2:
3:  #include <stdio.h>
4:  #include <math.h>
5:
6:  int main( void )
7:  {
8:
9:     double x;
10:
11:    printf("Enter a number: ");
12:    scanf( "%lf", &x);
13:
14:    printf("\n\nOriginal value: %lf", x);
15:
16:    printf("\nCeil: %lf", ceil(x));
17:    printf("\nFloor: %lf", floor(x));
18:    if( x >= 0 )
19:       printf("\nSquare root: %lf", sqrt(x) );
```

www.plcworld.cn


```
20:     else
21:        printf("\nNegative number" );
22:
23:     printf("\nCosine: %lf\n", cos(x));
24:     return 0;
25: }
```

该程序的运行情况如下：

```
Enter a number: 100.95

Original value: 100.950000
Ceil: 101.000000
Floor: 100.000000
Square root: 10.047388
Cosine: 0.913482
```

分析：该程序只使用了 C 语言标准库中的少数几个数学函数。第 12 行读取用户输入的数字，然后将其打印。接下来，将这个值传递给 4 个 C 语言库中的数学函数：ceil()、floor()、sqrt()和 cos()。注意，仅当用户输入的数字不为负时，才调用 sqrt()函数，因为根据定义，负数没有平方根。您可以在程序中添加函数，以检测它的功能。

19.2 处理时间

C 语言库中包含多个用于处理时间的函数。在 C 语言中，时间指的是日期和时间。根多时间函数的原型和定义都位于头文件 time.h 中。

19.2.1 时间的表示

C 语言中的时间函数用两种方式来表示时间。其中比较基本的一种表示方法是，从 1970 年 1 月 1 日午夜开始过去的秒数。负数表示在此之前的某个时刻。时间值被存储为 long 类型的整数。在 time.h 中，使用 typedef 语句将符号 time_t 和 clock_t 定义为 long。在时间函数中，使用这些符号，而不是 long。

第二种表示时间的方法是，将其划分为年、月、日等。对于这种表示时间的方式，时间函数使用结构 tm。在 time.h 中，该结构被定义为：

```
struct tm {
    int tm_sec;     // seconds after the minute - [0,59]
    int tm_min;     // minutes after the hour - [0,59]
    int tm_hour;    // hours since midnight - [0,23]
    int tm_mday;    // day of the month - [1,31]
    int tm_mon;     // months since January - [0,11]
    int tm_year;    // years since 1900
    int tm_wday;    // days since Sunday - [0,6]
    int tm_yday;    // days since January 1 - [0,365]
    int tm_isdst;   // daylight savings time flag
};
```

19.2.2 时间函数

本节介绍 C 语言库中各种用于处理时间的函数。前面指出过，时间指的是日期以及小时、分、秒。描述完时间函数后，将通过一个程序来演示它们的用法。

www.plcworld.cn

1. 获取当前时间

要获取系统内部时钟的当前时间，可以使用 time()函数。该函数的原型如下：

```
time_t time(time_t *timeptr);
```

前面介绍过，在 time.h 中，time_t 被定义为 long 的别名。函数 time()返回从 1970 年 1 月 1 日午夜开始过去的秒数。如果给它传递了一个不为 NULL 的指针，time()函数还会将这个值存储在该指针指向的 time_t 变量中。因此，要将当前时间存储在 time_t 变量 now 中，可以这样编写代码：

```
time_t now;

now = time(0);
```

也可以这样编写代码：

```
time_t now;
time_t *ptr_now = &now;
time(ptr_now);
```

2. 在不同的时间表示法之间进行转换

通常，知道从 1970 年 1 月 1 日起过去的秒数不太有用，因此 C 语言提供了一个 localtime()函数，用于将表示为 time_t 的时间转换为 tm 结构。tm 结构以更适合打印和显示的格式存储了年、月、日等时间信息。该函数的原型如下：

```
struct tm *localtime(time_t *ptr);
```

该函数返回一个指向静态结构 tm 的指针，因此使用该函数时，无需声明一个 tm 结构，而只需声明一个 tm 指针即可。每次调用 localtime()时，该静态结构都被重用和覆盖。如果要存储返回的值，则必须声明一个 tm 结构，并将静态结构的值复制到新声明的结构中。

函数 mktime()执行相反的转换：将 tm 结构转换为一个 time_t 值，其原型如下：

```
time_t mktime(struct tm *ntime);
```

该函数返回从 1970 年 1 月 1 日午夜开始，到指针 ntime 指向的 tm 结构表示的时间过去的秒数。

3. 显示时间

要将时间转换为适合显示的格式化字符串，可使用函数 ctime()和 asctime()。这两个函数都返回一个以特定格式表示时间的字符串。区别在于，前者接受以 time_t 表示的时间，而后者接受一个 tm 结构表示的时间。它们的原型如下：

```
char *asctime(struct tm *ptr);
char *ctime(time_t *ptr);
```

这两个函数都返回一个指针，该指针指向一个静态的、末尾不包含空字符的字符串，后者包含 26 个字符，以下面的格式表示传递给函数的时间：

```
Thu Jun 13 10:22:23 1991
```

时间采用 24 小时制。这两个函数都使用一个静态字符串，每次调用函数时，静态字符串都被重写。

要更好地控制时间的格式，可以使用 strftime()函数。该函数接受一个 tm 结构作为参数，并按格式化字符串对时间进行格式化。该函数的原型如下：

```
size_t strftime(char *s, size_t max, char *fmt, struct tm *ptr);
```

该函数按格式化字符串 fmt 指定的格式，对 ptr 指向的 tm 结构所表示的时间进行格式化，并将结果作为一个以空字符结尾的字符串写入到 s 指向的内存中。参数 max 表示 s 指向的内存空间大小。如果结果字符串包含的字符（包括结尾的空字符）超过 max 个，则函数返回 0，而 s 指向的字符串将是无效的。否则返回写入的字符数：strlen(s)。

格式化字符串包含表 19.5 中的 一个或多个转换说明符。

www.plcworld.cn

表 19.5　　可用于 strftime()中的转换说明符

说 明 符	替 换 为
%a	星期名（星期几）缩写
%A	完整的星期名（星期几）
%b	月份名缩写
%B	完整的月份名
%c	日期和时间表示（如 10:41:50 30-Jun-91）
%C	用 00～99 的十进制数字表示的年份
%d	用十进制数字 01～31 表示的日
%D	与"%m/%d/%y"等价
%e	用十进制数字 1～31 表示的日
%F	相当于“%Y-%m-%d”
%h	与“%b”相同，即月份名缩写
%H	用十进制数 00～23 表示的小时（24 小时制）
%I	用十进制数 00～11 表示的小时（12 小时制）
%j	用十进制数 001～366 表示的日
%m	用十进制数 01～12 表示的月份
%M	用十进制数 00～59 表示的分钟
%p	AM 或 PM
%r	当地时间（12 小时制）
%R	相当于“%H:%M”
%S	用十进制数 00～59 表示的秒
%T	相当于“%H:%M:%S”
%u	用十进制数 1～7 表示的星期几（1 表示星期一）
%U	用十进制数 00～53 表示的星期，每周从星期天开始。
%w	用十进制数 0～6 表示的星期几（0 表示星期天）
%W	用十进制数 00～53 表示的星期，每周从星期一开始。
%x	日期表示（如 30-Jun-91）
%X	时间表示（如 10:41:50）
%y	用十进制数 00～99 表示的年份
%Y	用十进制数表示的年份（四位）
%z	本地的时区（或缩写）。如果时区未知，则空着
%Z	时区名；如果未知，则空着
%%	一个百分符号

4. 计算时差

可以使用 difftime()宏来计算两个时间之间的差（单位为秒），该函数将两个 time_t 相减，并返回得到的差。该函数的原型如下：

```
double difftime(time_t later, time_t earlier);
```

该函数将 earlier 和 later 相减，并返回结果——两个时间之间相差的秒数。difftime()常用于计算过去的时间，如程序清单 19.2（其中还包含其他时间运算）所示。

可以使用 clock()函数计算代码运行的时间，该函数返回从程序开始执行起过去的时间，单位为 1/100 秒。该函数的原型如下：

```
clock_t clock(void);
```

www.plcworld.cn

要计算程序的某一部分执行的时间，可以调用 clock()两次——之前和之后分别调用一次，然后将两个返回的值相减。

19.2.3　使用时间函数

程序清单 19.2 演示了如何使用 C 语言库中的时间函数。

程序清单 19.2　　times.c：使用 C 语言库中的时间函数

```
1:  /* Demonstrates the time functions. */
2:
3:  #include <stdio.h>
4:  #include <time.h>
5:
6:  int main( void )
7:  {
8:      time_t start, finish, now;
9:      struct tm *ptr;
10:     char *c, buf1[80];
11:     double duration;
12:
13:     /* Record the time the program starts execution. */
14:
15:     start = time(0);
16:
17:     /* Record the current time, using the alternate method of */
18:     /* calling time(). */
19:
20:     time(&now);
21:
22:     /* Convert the time_t value into a type tm structure. */
23:
24:     ptr = localtime(&now);
25:
26:     /* Create and display a formatted string containing */
27:     /* the current time. */
28:
29:     c = asctime(ptr);
30:     puts(c);
31:     getc(stdin);
32:
33:     /* Now use the strftime() function to create several different */
34:     /* formatted versions of the time. */
35:
36:     strftime(buf1, 80, "This is week %U of the year %Y", ptr);
37:     puts(buf1);
38:     getc(stdin);
39:
40:     strftime(buf1, 80, "Today is %A, %x", ptr);
41:     puts(buf1);
42:     getc(stdin);
43:
44:     strftime(buf1, 80, "It is %M minutes past hour %I.", ptr);
```

www.plcworld.cn


```
45:     puts(buf1);
46:     getc(stdin);
47:
48:     /* Now get the current time and calculate program duration. */
49:
50:     finish = time(0);
51:     duration = difftime(finish, start);
52:   printf("\nProgram execution time using time() = %f seconds.", duration);
53:
54:     /* Also display program duration in hundredths of seconds */
55:     /* using clock(). */
56:
57:     printf("\nProgram execution time using clock() = %ld hundredths of sec.",
58:        clock());
59:     return 0;
60: }
```

该程序的输出如下：

```
Sun May 19 13:28:53 2002

This is week 20 of the year 2002

Today is Sunday, 05/19/02

It is 28 minutes past hour 01.

Program execution time using time() = 14.000000 seconds.
Program execution time using clock() = 14290 hundredths of sec.
```

分析：该程序包含多个注释行，因此应该很容易理解。由于使用了时间函数，因此第4行包含了头文件time.h。第8行声明了三个time_t变量：start、finish和now。这三个变量可用于存储从1970年1月1日起过去的时间（单位为秒）。第9行声明了一个指向tm结构的指针，tm结构在前面介绍过。其他变量的类型都是您熟悉的。

该程序的第15行记录程序开始执行时的时间，这是通过调用time()函数来实现的。然后该程序以不同的方式完成几乎相同的工作。第20行将一个指向变量now的指针传递给函数time()，而不是使用该函数的返回值。第24行完成第22行的注释指出的工作：将now的time_t值转换为一个tm结构。接下来的代码以各种不同的格式将当前的时间打印到屏幕上。第29行使用asctime()函数将信息赋给一个字符指针（c）。第30行打印格式化后的信息。然后，程序等待用户按下Enter键。

第36～46行使用strftime()函数，以三种不同的格式打印日期。根据表19.1可以知道这些代码行打印的内容。

然后程序再次获取当前的时间（第50行）——程序结束的时间。第51行使用difftime()函数，根据程序的结束时间和开始时间来计算程序运行了多长时间。第52行打印这个值。最后，该程序打印clock()函数计算的程序执行的时间。

19.3 处理错误

C语言标准库中包含了各种用于处理程序错误的函数和宏。

www.plcworld.cn

19.3.1　assert()宏

assert()宏可用于诊断程序 bug。该宏是在 assert.h 中定义的，其原型如下：

```
void assert(int expression);
```

其中参数 expression 可以是任何要检测的东西：变量或任何表达式。如果 expression 为 TRUE，则 assert() 不执行任何操作；否则在 stderr 上显示一条错误消息，并终止程序的执行。

如何使用 assert()呢？它最常见的用途是，用于查找程序 bug（不同于编译错误）。bug 不会阻止程序编译，但导致程序得到错误的结果或不能正常地运行（如死锁）。例如，您编写的一个金融分析程序偶尔可能给出错误的答案。您怀疑可能是由于变量 interest_rate 的值为负（不应该出现这样的情况）引起的。为验证您的怀疑，可以将下面的语句：

```
assert(interest_rate >= 0);
```

放置在程序使用 interest_rate 的地方。如果该变量的值为负，则 assert()宏将指出。然后您便可以检查相关的代码，找出导致问题的原因。

要了解 assert()宏的工作方式，请运行程序清单 19.3。如果您输入一个非零值，程序将显示这个值，然后正常结束；如果您输入零，assert()宏将强行终止异常程序。您看到的错误消息具体是什么取决于使用的编译器，下面是一个典型的例子：

```
Assertion failed: x, file list1903.c, line 13
```

为使 assert()发挥作用，编程程序是必须采用调试模式。有关如何启用调试模式（稍后介绍），请参阅编译器文档。以后以发行模式编译程序的最后版本时，assert()将被禁用。

程序清单 19.3　　assert.c：使用 assert()宏

```
1:  /* The assert() macro. */
2:
3:  #include <stdio.h>
4:  #include <assert.h>
5:
6:  int main( void )
7:  {
8:      int x;
9:
10:     printf("\nEnter an integer value: ");
11:     scanf("%d", &x);
12:
13:     assert(x >= 0);
14:
15:     printf("You entered %d.\n", x);
16:     return 0;
17: }
```

该程序的运行情况如下：

```
Enter an integer value: 10
You entered 10.
Enter an integer value: -1

Assertion failed: x, file list1903.c, line 13

Abnormal program termination
```

错误消息可能随系统和编译器而异，但基本思想相同。例如，如果使用的是 Dev-C++编译器，则用户输

www.plcworld.cn

入-1 时，输出将如下所示：

```
Enter an integer value: -1
c:\assert.c:13: failed assertion 'x >= 0'

This application has requested the Runtime to terminate it in an unusual
way.
Please contact the application's support team for more information.
```

分析：请运行该程序，看看第 13 行的 assert()显示的错误消息，其中包括未通过测试的表达式、程序的文件名以及 assert()所在代码行的行号。

assert()采取的动作取决于另一个名为 NDEBUG 宏（表示“不调试”）。如果 NDEBUG 宏未被定义（默认情况），则 assert()将处于活动状态；否则，assert()将被关闭，因此不起作用。如果您在程序的不同地方使用了 assert()来帮助调试，则解决问题后，可以定义 NDEBUG，将 assert()关闭。这样做比在程序中查找并删除 assert()语句要容易得多（后来您可能发现，还需要使用它们）。要定义 NDEBUG 宏，可使用编译指令#define。可以将下述代码行加入到程序清单 19.3 的第 2 行中，来查看结果：

```
#define NDEBUG
```

现在，即使您输入-1，程序也将打印这个值，并正常结束。

无需将 NDEBUG 定义为任何特定的值，而只需将其包含在编译指令#define 中即可。第 21 天的课程将更详细地介绍编译指令#define。

19.3.2 头文件 errno.h

头文件 errno.h 定义了几个宏，用于定义和记录运行错误。这些宏将结合函数 perror()一起使用，该函数将在下一节介绍。

头文件 errno.h 定义了一个名为 errno 的外部 int 变量。如果执行时发生错误，C 语言库中的很多函数都将一个值赋给该变量。头文件 errno.h 还定义了一组符号常量，用于表示其他错误，如表 19.6 所示。

表 19.6　　头文件 errno.h 定义的符号错误常量

名　称	值	消息和含义
E2BIG	1000	参数列表过长（超过 128 字节）
EACCES	5	没有权限（例如，试图写一个为只读而打开的文件）
EBADF	6	文件描述符无效
EDOM	1002	数学参数超出范围（将一个不允许的值作为参数传递给数学函数）
EEXIST	80	文件已经存在
EMFILE	4	打开过多的文件
ENOENT	2	没有这样的文件或目录
ENOEXEC	1001	执行格式错误
ENOMEM	8	内存不够（例如，没有足够的内存来执行 exec()函数）
ENOPATH	3	路径未找到
ERANGE	1003	结果超出范围（例如，相对于返回数据类型而言，数学函数返回的值过大或过小）

可以以两种方式来使用 errno。有些函数通过其返回值来指出发生了错误。在这种情况下，您可以通过检测 errno 的值来确定错误的性质，并采取相应的措施。如果无法知道是否发生了错误，可以检测 errno，如果它不为零，则说明发生了错误，而 errno 的值指出了错误的性质。处理完错误后，一定要将 errno 的值重置为零。下一节将介绍 perror()函数，然后显示 errno 的用法（程序清单 19.4）。

19.3.3 perror()函数

perror()函数是 C 语言中的另一个错误处理工具，该函数在 stderr 上显示一条消息，指出库函数调用或系

www.plcworld.cn

统调用期间，最后发生的一个错误。该函数的原型如下，它位于头文件 stdio.h 中：

```
vcid perror(const char *msg);
```

其中参数 msg 指向一条可选的、用户定义的消息。该函数首先打印这条消息，然后是冒号以及实现定义的、描述最后发生的错误的消息。如果 perror()被调用时没有发生错误，则显示的消息为 no error。

perror()不会根据错误来采取某种措施，要采取措施，您必须在程序中编写相应的代码。这种措施可能是提供用户执行某种操作，如终止程序。程序可以检测 errno 的值，并根据错误的性质采取相应的措施。使用外部变量 errno 时，程序无需包含头文件 errno.h；而只有需要使用表 19.5 列出的符号错误常量时，才必须包含这个头文件。程序清单 19.4 演示了如何使用 perror()和 errno 来处理运行错误。

程序清单 19.4　　perror.c：使用 perror()和 errno 处理运行错误

```
1:  /* Demonstration of error handling with perror() and errno. */
2:
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:  #include <errno.h>
6:
7:  int main( void )
8:  {
9:     FILE *fp;
10:    char filename[80];
11:
12:    printf("Enter filename: ");
13:    gets(filename);
14:
15:    if (( fp = fopen(filename, "r")) == NULL)
16:    {
17:       perror("You goofed!");
18:       printf("errno = %d.\n", errno);
19:       exit(1);
20:    }
21:    else
22:    {
23:       puts("File opened for reading.");
24:       fclose(fp);
25:    }
26:    return 0;
27: }
```

该程序的运行情况如下：

```
Enter file name: perror.c
File opened for reading.

Enter file name: notafile.xxx
You goofed!: No such file or directory
errno = 2.
```

分析：该程序根据是否可以打开文件进行读取，打印两条消息中的一条。第 15 行试图打开一个文件。如果打开了，则执行 if 语句的 else 部分，打印下述消息：

```
File opened for reading.
```

如果打开文件时发生错误（如文件不存在），则执行 if 语句中的第 17～19 行。第 17 行调用 perror()函数，并将字符串“You goofed!”传递给它。然后打印错误号。如果输入一个不存在的文件名，则程序的输出如下：

www.plcworld.cn


```
Ycu goofed!: No such file or directory.
errno = 2
```

应　该	不 应 该
一定要在程序中检查可能发生的错误，而不要认为一切都会正常进行。	无需使用表 19.5 中的符号错误常量时，不要包含头文件 errno.h。

19.4　查找和排序

查找和排序是程序经常需要执行的两种任务。C 语言标准库包含了可用于完成这些任务的通用函数。

19.4.1　使用 bsearch()进行查找

库函数 bsearch()采用二分法，在数组中查找一个与键值（key）匹配的元素。要使用 bsearch()，数组中的元素必须按升序排列。另外，还必须提供比较函数，供 bsearch()用来确定一个数据项是大于、等于还是小于另一个数据项。bsearch()的原型如下，它位于头文件 stdlib.h 中：

```
void *bsearch(const void *key, const void *base, size_t num, size_t width,
int (*cmp)(const void *element1, const void *element2));
```

这个函数原型很复杂，有必要对其做详细介绍。参数 key 是一个指针，指向要查找的数据项；而 base 也是一个指针，它指向要在其中进行查找的数组的第一个元素。这两个指针都是 void 类型，因此可以指向任何数据对象。限定符 const 指出传递的值将被视为常量，函数不能修改它们。

参数 num 指出数组包含的元素数，而 width（单位为字节）是每个元素的长度。类型说明符 size_t 指的是 sizeof()运算符返回的数据类型，相当于 unsigned。通常使用 sizeof()运算符来计算得到 num 和 width 的值。

最后，参数 cmp 是一个指向比较函数的指针。比较函数可以是用户编写的函数，也可以是库函数 strcmp()（查找字符串时）。比较函数必须满足下述要求：

- 接受两个指向数据项的指针作为参数；
- 返回这样的一个 int 值：

a. 如果元素 1 小于元素 2，则小于 0；

b. 如果元素 1 等于元素 2，则为 0；

c. 如果元素 1 大于元素 2，则小于 0。

bsearch()函数返回一个 void 指针。该指针指向第一个与键值匹配的数组元素；如果没有找到匹配的元素，则为 NULL。使用返回的指针之前，必须将其强制转换为合适的类型。

可以使用 sizeof()运算符来指定参数 num 和 width 的值。如果被查找的数组名为 array，则下面的语句：

```
sizeof(array[0]);
```

将返回 width 的值：一个数组元素的长度（单位为字节）。由于表达式 sizeof(array)返回整个数组的长度（单位为字节），因此可以使用下面的语句来计算 num 的值——数组包含的元素数：

```
sizeof(array)/sizecf(array[0])
```

二分法查找算法的效率非常高，可以快速地查找大型数组。它要求数组元素按升序排列。这种算法的工作原理如下：

1. 将键值与数组正中间的元素进行比较。如果它们匹配，则结束查找；否则，键值要么大于该数组元素，要么比它小。

2. 如果键值小于该数组元素，则如果存在匹配的数组元素，它可定位于数组的前半部分；如果键值大于该数组元素，则匹配的元素肯定位于数组的后半部分；

3. 根据上述结果确定要在数组的哪一部分查找，然后返回第 1 步。

二分法查找每进行一次比较，便可排除被查找数组的一半元素。例如，对于包含 1000 个元素的数组，只需进行 10 次比较便可完成查找；而对于包含 16000 个元素的数组，只需进行 14 次比较。总之，对包含 2^n 个

www.plcworld.cn

元素的数组进行查找时，只需比较 n 次。

19.4.2　使用 qsort()进行排序

库函数 qsort()实现的是快速排序法，这种方法是 C.A.R. Hoare 发明的。该函数对数组中的元素进行排序。通常结果是按升序排列的，但也可按降序排列。该函数的原型如下，它是在头文件 stdlib.h 中定义的：

```
void qsort(void *base, size_t num, size_t size,
int (*cmp)(const void *element1, const void *element2));
```

其中参数 base 是一个指向数组第一个元素的指针，num 是数组包含的元素数，size 是数组元素的长度（单位为字节）。参数 cmp 是一个指向比较函数的指针。比较函数必须满足的条件与前一节介绍的 bsearch()函数使用的比较函数相同：您通常在 bsearch()和 qsort()函数中使用同一个比较函数。函数 qsort()没有返回值。

19.4.3　演示查找和排序

程序清单 19.5 演示了如何使用 qsort()和 bsearch()函数。该函数首先对数组元素进行排序，然后在其中进行查找。这里使用了非 ANSI 函数 getch()。如果您的编译器不支持它，可以使用 ANSI 标准函数 getchar()替换它。

程序清单 19.5　　sort.c：使用 qsort()和 bsearch()函数对数值进行排序和查找

```
1:  /* Using qsort() and bsearch() with values.*/
2:
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:
6:  #define MAX 20
7:
8:  int intcmp(const void *v1, const void *v2);
9:
10: int main( void )
11: {
12:     int arr[MAX], count, key, *ptr;
13:
14:     /* Enter some integers from the user. */
15:
16:     printf("Enter %d integer values; press Enter after each.\n", MAX);
17:
18:     for (count = 0; count < MAX; count++)
19:        scanf("%d", &arr[count]);
20:
21:     puts("Press Enter to sort the values.");
22:     getc(stdin);
23:
24:     /* Sort the array into ascending order. */
25:
26:     qsort(arr, MAX, sizeof(arr[0]), intcmp);
27:
28:     /* Display the sorted array. */
29:
30:     for (count = 0; count < MAX; count++)
31:        printf("\narr[%d] = %d.", count, arr[count]);
32:
33:     puts("\nPress Enter to continue.");
```

www.plcworld.cn


```
34:     getc(stdin);
35:
36:     /* Enter a search key. */
37:
38:     printf("Enter a value to search for: ");
39:     scanf("%d", &key);
40:
41:     /* Perform the search. */
42:
43:     ptr = (int *)bsearch(&key, arr, MAX, sizeof(arr[0]),intcmp);
44:
45:     if ( ptr != NULL )
46:        printf("%d found at arr[%d].", key, (ptr - arr));
47:     else
48:        printf("%d not found.", key);
49:     return 0;
50: }
51:
52: int intcmp(const void *v1, const void *v2)
53: {
54:     return (*(int *)v1 - *(int *)v2);
55: }
```

该程序的运行情况如下：

```
Enter 20 integer values; press Enter after each.
45
12
999
1000
321
123
2300
954
1968
12
2
1999
1776
1812
1456
1
9999
3
76
200
Press Enter to sort the values.

arr[0] = 1.
arr[1] = 2.
arr[2] = 3.
arr[3] = 12.
arr[4] = 12.
```

www.plcworld.cn


```
arr[5] = 45.
arr[6] = 76.
arr[7] = 123.
arr[8] = 200.
arr[9] = 321.
arr[10] = 954.
arr[11] = 999.
arr[12] = 1000.
arr[13] = 1456.
arr[14] = 1776.
arr[15] = 1812.
arr[16] = 1968.
arr[17] = 1999.
arr[18] = 2300.
arr[19] = 9999.
Press Enter to continue.

Enter a value to search for:
1776
1776 found at arr[14]
```

分析：该程序应用了前面介绍的有关排序和查找的所有知识。它让用户输入 MAX（这里为 20）个值，对它们进行排序，并按顺序打印它们。然后程序让您输入一个要在数组中查找的值，并打印一条消息，指出查找结果。

第 18 和 19 行使用您熟悉的代码来读取用户输入的值。第 26 行调用 qsort()对数组进行排序。其中第一个参数是指向第一个数组元素的指针；接下来的参数为 MAX（数组包含的元素数）；然后是第一个元素的长度（让 qsort()函数知道每个数据项的宽度）；最后的一个参数是 intcmp。

函数 intcmp 是在第 52～55 行定义的。它返回传递给它的两个值之间的差。乍一看，这很简单，但别忘了比较函数应该返回的值：如果元素相等，则返回 0；如果第一个元素大于第二个，则返回一个正值；如果第一个元素小于第二个，则返回一个负值。intcmp()函数正是这样的。

查找是使用 bsearch()来完成的。该函数的参数与 qsort()几乎完全相同，差别在于，bsearch()的第一个参数是要查找的键值。bsearch()返回一个指针，该指针指向匹配的元素；如果没有找到这样的元素，则为 NULL。第 43 行将 bsearch()返回的值赋给 ptr，然后第 45～48 行根据 ptr 的值打印查找结果。

程序清单 19.6 的功能与程序清单 19.5 相同，只是对字符串进行排序和查找。

程序清单 19.6　　strsort.c：使用 qsort()和 bsearch()对字符串进行排序和查找

```
1:  /* Using qsort() and bsearch() with strings. */
2:
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:  #include <string.h>
6:
7:  #define MAX 20
8:
9:  int comp(const void *s1, const void *s2);
10:
11: int main( void )
12: {
13:     char *data[MAX], buf[80], *ptr, *key, **key1;
14:     int count;
15:
```

www.plcworld.cn


```
    /* Input a list of words. */

    printf("Enter %d words, pressing Enter after each.\n",MAX);

    for (count = 0; count < MAX; count++)
    {
        printf("Word %d: ", count+1);
        gets(buf);
        data[count] = malloc(strlen(buf)+1);
        strcpy(data[count], buf);
    }

    /* Sort the words (actually, sort the pointers). */

    qsort(data, MAX, sizeof(data[0]), comp);

    /* Display the sorted words. */

    for (count = 0; count < MAX; count++)
        printf("\n%d: %s", count+1, data[count]);

    /* Get a search key. */

    printf("\n\nEnter a search key: ");
    gets(buf);

    /* Perform the search. First, make key1 a pointer */
    /* to the pointer to the search key.*/

    key = buf;
    key1 = &key;
    ptr = bsearch(key1, data, MAX, sizeof(data[0]), comp);

    if (ptr != NULL)
        printf("%s found.\n", buf);
    else
        printf("%s not found.\n", buf);
    return 0;
}

int comp(const void *s1, const void *s2)
{
    return (strcmp(*(char **)s1, *(char **)s2));
}
```

该程序的运行情况如下：

```
Enter 20 words, pressing Enter after each.
Word 1: apple
Word 2: orange
Word 3: grapefruit
Word 4: peach
Word 5: plum
```

www.plcworld.cn


```
Word 6: pear
Word 7: cherries
Word 8: banana
Word 9: lime
Word 10: lemon
Word 11: tangerine
Word 12: star
Word 13: watermelon
Word 14: cantaloupe
Word 15: musk melon
Word 16: strawberry
Word 17: blackberry
Word 18: blueberry
Word 19: grape
Word 20: cranberry

1: apple
2: banana
3: blackberry
4: blueberry
5: cantaloupe
6: cherries
7: cranberry
8: grape
9: grapefruit
10: lemon
11: lime
12: musk melon
13: orange
14: peach
15: pear
16: plum
17: star
18: strawberry
19: tangerine
20: watermelon

Enter a search key: orange
orange found.
```

分析：对于程序清单 19.6，有必要说明的有两点。该程序使用了一个字符串指针数组——第 15 天的课程中介绍的一种技术。正如该章介绍过的，可以对指针数组进行排序，从而实现对字符串的排序。然而，采用这种方法时，需要对比较函数进行修改，使之接受两个指针参数（它们指向要比较的数组元素）。然而，您希望根据指针指向的字符串的值，而不是指针本身的值对指针进行排序。

因此，比较函数必须接受指向指针的指针作为参数。comp()的每个参数都是指向数组元素的指针，而每个元素本身又是一个指向字符串的指针，因此这些参数都是指向指针的指针。在函数中，需要对指针执行解除引用的操作，使得 comp()的返回值取决于被指向的字符串的值。

由于传递给 comp()的参数是指向指针的指针，这导致了另一个问题。要查找的键值被存储在 buf[]中，而数组名（这里为 buf）是指向数组的指针。然而，您需要传递的不是 buf 本身，而是指向它的指针。问题是，buf 是一个指针常量，而不是指针变量。buf 本身没有内存地址，而是一个符号，其值为数组的地址。因此，您不能通过在 buf 前加上地址运算符（即&buf）来创建一个指向 buf 的指针。

www.plcworld.cn

那么，怎么办呢？首先，创建一个指针变量，并将 buf 的值赋给它。在上述程序中，该指针变量名为 key。由于 key 是一个指针变量，因此有地址，所以可以创建一个值为该地址的指针（这里为 key1）。最后调用 bsearch() 时，第一个参数为 key1（它是一个指向指针的指针，它指向的指针指向键值字符串）。然后，函数 bsearch() 将该参数再传递给 comp()，这样一切都将正常运行。

应　读	不应读
一定要查阅编译器文档或 ANSI 文档，以获悉您可以使用的其他标准函数	使用 bsearch()之前，别忘了将数组按升序排列

19.5 总　结

今天的课程介绍了 C 语言函数库中一些比较有用的函数，包括执行数学计算、处理时间和处理错误的函数。用于对数据进行查找和排序的函数很有用，可以帮助您节省大量的编程时间。

19.6 问与答

问：为什么几乎所有的数学函数都返回 double 值？

答：旨在提高精度，而不是为了统一。double 比其他变量类型的精度高，因此结果将更精确。第 20 天的课程将介绍有关强制类型转换和提升（promotion）的细节，这些主题适用于精度。

问：在 C 语言中进行排序和查找时，只能使用 qsort()和 bsearch()函数吗？

答：这些函数是由标准库提供的，但您不一定非得使用它们。很多有关计算机编程的图书介绍如何编写查找和排序程序。C 语言提供了编写这种程序所需的所有命令，您也可以购买编写好的查找和排序例程。bsearch()和 qsort()的最大优势在于，它们是现成的，并且任何 ANSI-兼容编译器都提供了它们。

问：数学函数会检查参数的合法性吗？

答：决不要假设输入的值是正确的，一定要验证用户输入的值是否有效。例如，如果您把一个负值传递给 sqrt()，函数将引发错误。如果您要格式化输出，则不希望这种错误消息按原来的样子显示。请删除程序清单 19.1 中的 if 语句，然后输入一个负数，以理解上一句话的含义。

19.7 作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

19.7.1 小测验

1. C 语言中的所有数学函数都返回什么类型？
2. time_t 类型是什么类型的别名？
3. time()函数和 clock()函数之间的区别何在？
4. 当您调用 perror()函数时，它会纠正错误条件吗？
5. 使用 bsearch()在数组中进行查找之前，必须如何做？
6. 要使用 bsearch()在包含 16000 个元素的数组中查找一个元素，需要进行多少次比较？
7. 要使用 bsearch()在包含 10 个元素的数组中查找一个元素，需要进行多少次比较？
8. 要使用 bsearch()在包含两百万个元素的数组中查找一个元素，需要进行多少次比较？
9. 对于供 bsearch()和 qsort()使用的比较函数，其返回值应满足什么样的要求？

www.plcworld.cn

10. 如果没有在数组中找到匹配的元素，bsearch()将返回什么？

19.7.2 练习

1. 编写调用 bsearch()的代码，要查找的数组名为 names，其元素为字符串。使用的比较函数名为 comp_names()。假设数组中所有元素的长度都相同。

2. 排错：下述程序有何错误？

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
      int values[10], count, key, *ptr;

      printf("Enter values");
      for( ctr = 0; ctr < 10; ctr++ )
           scanf( "%d", &values[ctr] );
        qsort(values, 10, compare_function());
}
```

3. 排错：下述比较函数有错误吗？

```
int intcmp( int element1, int element2)
{
   if ( element 1 > element 2 )
      return -1;
   else if ( element 1 < element2 )
      return 1;
   else
      return 0;
}
```

附录 F 没有提供下述练习的答案。

4. 选做题：修改程序清单 19.1，使 sqrt()能够处理负数，方法是把 x 的绝对值传递给该函数。

5. 选做题：编写一个程序，它提供一个菜单来执行各种数学函数。请包含尽可能多的数学函数。

6. 选做题：使用今天介绍的时间函数，编写一个让程序暂停大约 5 秒钟的函数。

7. 选做题：在练习 4 的程序中添加 assert()函数。当用户输入的是负数时，打印一条消息。

8. 选做题：编写一个程序，让用户输入 30 个姓名，并使用 qsort()对它们进行排序，然后打印排序后的姓名。

9. 选做题：修改练习 8 中编写的程序，以便当用户输入 QUIT 时，程序停止读取姓名，并对输入的姓名进行排序。

10. 选做题：参考第 15 天的课程中根据指针指向的字符串的值，对指针进行排序的方法，编写一个程序，计算使用这种方法对一个大型指针数组进行排序所需的时间，并将其与使用库函数 qsort()完成同样任务所需的时间进行比较。

www.plcworld.cn

TYPE & RUN 6　计算抵押贷款的偿还金额

顾名思义，该 Type & Run 抵押贷款或其他任何贷款的偿还金额。该程序提示用户输入下述三项信息：

- 贷款金额：要借贷多少钱（也叫本金）；
- 年利率：每年收取的利息。用户应输入实际的利率，例如，利率为 8.5%时，应输入 8.5，而不要将其转换为实际的数值（这里为 0.085），因为程序将为您完成这种转换。
- 贷款期限（单位为月）：即多少个月后，您将还请贷款。

输入并运行该程序时，可以计算抵押贷款和其他贷款的偿还金额。

程序清单 R&T 6　　mortgage.c：计算抵押贷款的偿还金额

```
1:  /* Calculates loan/mortgage payments. */
2:
3:  #include <stdio.h>
4:  #include <math.h>
5:  #include <stdlib.h>
6:
7:   int main( void )
8:  {
9:     float principal, rate, payment;
10:    int term;
11:    char ch;
12:
13:    while (1)
14:    {
15:       /* Get loan data */
16:       puts("\nEnter the loan amount: ");
17:       scanf("%f", &principal);
18:       puts("\nEnter the annual interest rate: ");
19:       scanf("%f", &rate);
20:       /* Adjust for percent. */
21:       rate /= 100;
22:       /* Adjust for monthly interest rate. */
23:       rate /= 12;
24:
25:       puts("\nEnter the loan duration in months: ");
26:       scanf("%d", &term);
27:       payment = (principal * rate) / (1 - pow((1 + rate), -term));
28:       printf("Your monthly payment will be $%.2f.\n", payment);
29:
```

www.plcworld.cn


```
30:         puts("Do another (y or n)?");
31:         do
32:         {
33:         ch = getchar();
34:         } while (ch != 'n' && ch != 'y');
35:
36:         if (ch == 'n')
37:             break;
38:     }
39:     return 0;
40: }
```

分析：该程序假设是一种标准贷款，如固定利率的汽车贷款或房屋贷款。计算偿还金额时，使用下面的标准金融公式：

payment = (*P* * *R*) / (1 - (1 + *R*)^(-*T*))

其中 *P* 为本金，*R* 是利率，而 *T* 为贷款期限。符号^表示幂运算。在该公式中，利率和期限的计算单位必须相同，也就是说，如果贷款期限的单位为月，则利率必须是月利率。由于利率通常是年利率，因此第 23 行将年利率除以 12 得到月利率。偿还金额的计算是在第 27 行完成的，第 28 行打印计算结果。

www.plcworld.cn

第 20 天课程　管理内存

今天的课程介绍一些有关在 C 程序中管理内存的高级主题，包括以下内容：

- 类型转换；
- 如何分配和释放内存？
- 如何操纵内存块？
- 如何操纵位？

20.1　类型转换

在 C 语言中，所有的数据对象都有特定的类型。数值变量可能是 int 或 float 类型，指针可能指向 double 或 char 变量，等等。程序经常需要在同一个表达式或语句中使用不同的数据类型，在这种情况下，将发生什么？有时候，C 编译器自动处理不同的类型，您无需操心。而其他时候，您必须显式地将一种数据类型转换为另一种，以防止结果不正确。在前面的课程中，您遇到过这样的情况，即使用 void 指针之前，必须将其转换为特定的类型。在这种情况下，您要知道何时必须显式地进行类型转换，如果不进行合适的转换，将发生什么样的错误。接下来的几节介绍自动类型转换和显式类型转换。

20.1.1　自动类型转换

顾名思义，自动类型转换是由编译器自动完成的，而无需程序员的干预。然而，您必须知道发生的情况，从而了解编译器如何计算表达式。



注意：自动类型转换常被称为隐式转换。

1．类型提升

表达式被计算时，结果值将为一种特定的数据类型。如果表达式中的所有组成部分的类型都相同，则结果也将是这样的类型。例如，如果 x 和 y 的类型都为 int，则下述表达式的类型也为 int：

```
x + y
```

如果表达式中各个组成部分的数据类型不同，情况将如何呢？在这种情况下，表达式的类型与综合性最高的部分相同。对于数值类型，综合性从低到高的排列顺序如下：

```
char
short
int
long
long long
float
```

www.plcworld.cn


```
double
long double
```

因此，如果表达式包含类型为 int 和 char 的组成部分，则其本身的类型为 int；如果表达式包含类型为 long 和 float 的组成部分，则其本身的类型为 float；等等。

创建表达式时，编译器每次使用两个变量或值。例如，对于下面的表达式：

```
Y + X * 2
```

编译器将首先使用操作数 X 和 2。计算完毕后，再使用计算结果和操作数 Y。

必要时，表达式中的操作数将被提升，以便同与之相关联的操作数匹配。在表达式中，对每个双目运算符两边的操作数进行提升。当然，如果两个操作数的类型相同，则不提升。如果不相同，则按下面的规则进行提升：

- 如果其中一个操作数的类型为 long double，则将另一个提升为 long double 类型；
- 如果其中一个操作数的类型为 double，则将另一个提升为 double 类型；
- 如果其中一个操作数的类型为 float，则将另一个提升为 float 类型；
- 如果其中一个操作数的类型为 long，则将另一个提升为 long 类型。

例如，如果 x 的类型为 int，而 y 的类型为 float，则在计算表达式 x/y 时，将把 x 提升为 float 类型，然后再计算该表达式的值。这并不意味着变量 x 的类型被修改，而意味着创建 x 的一个拷贝，并在计算表达式时使用该拷贝。表达式的结果的类型为 float。同样，如果 x 的类型为 double，而 y 的类型为 float，则 y 将被提升为 double 类型。

2. 赋值时的转换

赋值时，也会进行提升。赋值语句右边的表达式总是被提升为赋值运算符左边的数据对象的类型。注意，这可能导致“降级”，而不是提升。如果 f 的类型为 float，而 i 的类型为 int，则在下面的赋值语句中，i 将被提升为 float 类型：

```
f = i;
```

而在下面的赋值语句中：

```
i = f;
```

f 将被降级为 int 类型，其小数部分将被丢弃。别忘了，f 本身并没有任何变化，提升影响的只是它的拷贝。因此，执行下述语句后：

```
float f = 1.23;
int i;
i = f;
```

变量 i 的值为 1，而 f 的值仍为 1.23。这个例子表明，将浮点数转换为 int 类型时，其小数部分丢失。



注意：当变量被隐式地降级时，大多数编译器会发出警告。

您需要明白的是，当一个 int 值被转换为 float 类型时，得到的浮点数可能与原来的整型值不完全相同。这是因为计算机内部使用的浮点格式无法精确地表示所有的整数。例如，下面的代码可能显示 2.999995，而不是 3：

```
float f;
int i = 3;
f = i;
printf("%f", f);
```

在大多数情况下，导致的误差都是微不足道的。然而，一定要将整数值存储在 short、int、long 或 long long 类型的变量中。

www.plcworld.cn

20.1.2 显式转换

强制类型转换（typecast）使用转换运算符（cast operator）来显式地控制类型转换。强制类型转换由位于表达式前面、用圆括号括起的类型名组成。可以对算术表达式和指针执行强制类型转换。结果是，表达式被转换为指定的类型。通过这种方式，可以控制表达式的类型，而不依赖于编译器的自动转换。

1. 强制转换算术表达式的类型

强制转换算术表达式命令编译器以特定的方式表示该表达式的值。强制转换的效果类似于前面讨论的提升，但这是由程序员而不是编译器控制的。例如，如果 i 的类型为 int，则下述表达式将 i 的类型转换为 float：

```
(float) i
```

换句话说，程序将创建 i 的一个拷贝，其格式为浮点数。

何时需要强制转换算术表达式的类型呢？最常见的情况是，避免执行整数除法运算时丢失结果的小数部分。程序清单 20.1 说明了这一点。您应编译并运行该程序。

程序清单 20.1　　casting.c：两个整数相除时，结果的小数部分将丢失

```
1:  #include <stdio.h>
2:
3:  int main( void )
4:  {
5:     int i1 = 100, i2 = 40;
6:     float f1;
7:
8:     f1 = i1/i2;
9:     printf("%lf\n", f1);
10:    return 0;
11: }
```

该程序的输出如下：

```
2.000000
```

分析：该程序显示的结果为 2.000000，但 100/40 的结果应为 2.5。出现了什么问题呢？第 8 行中的表达式 i1/i2 包含两个 int 变量。根据前面介绍的规则可知，表达式 i1/i2 的值为 int 类型，因为两个操作数的类型都是 int。因此结果中只包含整数部分，小数部分丢失了。

您可能认为，把 i1/i2 的结果赋给一个 float 变量将把它提升为 float 类型。确实是这样，但这已经太晚了，此时结果的小数部分已经被丢弃。

为避免这种误差，必须将其中的一个 int 变量强制转换为 float 类型。这样，根据前面介绍的规则，另一个变量将被自动提升为 float 类型，因此表达式的值也为 float 类型，从而结果的小数部分得以保留。为说明这一点，可将上述源代码的第 8 行修改为如下所示：

```
f1 = (float)i1/i2;
```

这样程序将显示正确的答案。



> 注意：在更复杂的表达式中，可能需要强制转换多个值的类型。

2. 强制转换指针的类型

您已经见过强制转换指针类型的情况。在第 18 天的课程中，void 指针是一个通用指针，可以指向任何东西。使用这种指针之前，必须将其转换为合适的类型。请注意，给 void 指针赋值或将其同 NULL 进行比较时，

www.plcworld.cn

无需转换其类型；然而对其执行解除引用操作或指针算术之前，必须将其转换为合适的类型。有关强制转换指针类型的更详细的信息，请参阅第 18 天的课程。

应　该	不应该
必要时，应通过强制类型转换对变量的值进行提升或降级。	不要仅仅因为编译器发出警告而采取强制类型转换。您可能发现，采取强制类型转换消除了警告，但采用这种方式消除警告之前，一定要理解其中的原因。

20.2　分配内存的存储空间

C 语言库提供了用于在运行时分配内存空间（这被称为动态内存分配）的函数。与通过在程序源代码中声明变量、结构和数组来显式地分配内存相比，这种技术有明显的优势。前一种方法被称为静态内存分配，它要求您在编写程序时就知道需要多少内存。动态内存分配让程序能够在运行期间对内存需求（如用户输入）做出反应。要使用任何用于动态分配内存的函数，都必须包含头文件 stdlib.h，对于某些编译器，还需要包含头文件 malloc.h。所有的内存分配函数都返回一个 void 指针。正如第 18 天的课程介绍的，使用 void 指针之前，必须将其转换为合适的类型。

介绍细节之前，先简要地介绍一下内存分配。内存分配是什么意思呢？每台计算机都安装了一定的内存（随机存储器，RAM）。内存量随系统而异。当您运行程序时，无论它是字处理器、图形程序还是您编写的 C 程序，它都将被从磁盘装入到计算机内存中。程序占用的内存空间包括程序代码和程序的静态数据（源代码中声明的数据项）占用的空间。余下的空间则可供本节介绍的函数进行分配。

可供分配的内存有多少呢？完全不确定。如果您在系统中运行一个大型程序，而该系统安装的内存量适中，则余下的可用内存就很少。相反，如果在一个安装了大量内存的系统上只运行一个小型程序，则可用的内存就很多。这意味着程序不能对可用内存量做任何假设，因此调用内存分配函数时，必须检查它的返回值，确保成功地分配了内存。另外，程序还必须妥善地处理内存分配失败的情况。本章的后面将介绍一种用于确定还有多少可用内存的技术。

另外，操作系统也会影响可用的内存量。有些操作系统只允许使用一部分物理 RAM。DOS 6.x 及更早的版本就属于这种类型。即使您的系统安装了很多内存，DOS 程序也只能直接存取前 640KB 内存（可以采用特殊的技术来存取其他内存，但这已超出了本书的范围）。然而，UNIX 系统通常允许程序使用所有的物理内存。另外，有些操作系统，如 Windows 和 OS/2，还提供了虚拟内存，允许分配硬盘上的存储空间，就像它是 RAM 一样。在这种情况下，程序可用的内存不但包括 RAM，还包括硬盘上的虚拟内存空间。

对您而言，操作系统在内存分配上的差异在很大程度上是透明的。使用 C 函数来分配内存时，它要么成功，要么失败，而您无需考虑到底发生了什么。

20.2.1　使用 malloc()函数分配内存

在以前的课程中，已经介绍过如何使用库函数 malloc()来为字符串分配内存空间。malloc()函数不仅可用于为字符串分配空间，还可用于为任何存储类型分配空间。该函数分配空间时，分配的最小单位为字节。malloc()的原型如下：

```
void *malloc(size_t num);
```

其中 size_t 是在 stdlib.h 中定义的，它相当于 unsigned。malloc()函数分配 num 个字节的内存空间，并返回指向第一个字节的指针。如果无法分配请求的内存空间或者 num 的值为 0，则该函数返回 NULL。如果您还不明白该函数的工作原理，请复习第 10 天课程的“malloc()函数”一节。

程序清单 20.2 演示了如何使用 malloc()来确定系统中可用的内存量。在 DOS 系统或 Windows 旧版本的 DOS 模式下，该程序能够正常运行。但要注意的是，在新的 Windows 版本、OS/2 和 UNIX 系统（这些系统使用硬盘空间来提供虚拟内存）下运行该程序时，结果将是奇异的。为耗尽可用内存，该程序可能需要花很长时间。

www.plcworld.cn



警告：如果您的操作系统允许同时运行多个程序，请在运行程序清单20.2中的程序之前，退出其他所有的程序。如果您耗尽了系统中的所有内存，系统可能死锁或出现其他不可预测的情况。

程序清单20.2　　malloc.c：使用malloc()确定可用的内存量

```
1:  /* Using malloc() to determine free memory.*/
2:
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:
6:  /* Definition of a structure that is
7:     1024 bytes (1 kilobyte) in size.) */
8:
9:  struct kilo {
10:    struct kilo *next;
11:    char dummy[1020];
12: };
13:
14: int FreeMem(void);
15:
16: int main( void )
17: {
18:
19:    printf("You have %d kilobytes free.\n", FreeMem());
20:    return 0;
21: }
22:
23: int FreeMem(void)
24: {
25:    /*Returns the number of kilobytes (1024 bytes)
26:    of free memory. */
27:
28:    long counter;
29:    struct kilo *head, *current, *nextone;
30:
31:    current = head = (struct kilo*) malloc(sizeof(struct kilo));
32:
33:    if (head == NULL)
34:       return 0;      /*No memory available.*/
35:
36:    counter = 0;
37:    do
38:    {
39:       counter++;
40:       current->next = (struct kilo*) malloc(sizeof(struct kilo));
41:       current = current->next;
42:       printf("\r%d", counter);
43:    } while (current != NULL);
44:
45:    /* Now counter holds the number of type kilo
```

www.plcworld.cn


```
46:       structures we were able to allocate. We
47:       must free them all before returning. */
48:
49:    current = head;
50:    do
51:    {
52:       nextone = current->next;
53:       free(current);
54:       current = nextone;
55:    } while (nextone != NULL);
56:
57:    return counter;
58: }
```

该程序的输出如下：

```
You have 60 kilobytes free.
```

分析：该程序不断分配内存块，直到 malloc()函数返回 NULL（表明没有可用的内存）为止。在安装了大量内存的系统上运行该程序时，可能需要几分钟才会结束。第 42 行的打印语句显示计数器，让用户知道程序在运行。

将分配的内存块数与内存块的大小相乘，得到可用内存量。然后函数释放分配的所有内存，并将分配的内存块数返回给调用程序。由于内存块的大小为 1KB，因此返回的值直接指出了可用内存量（单位为 KB）。您可能知道，1KB 并非 1000 字节，而是 1024（2 的 10 次方）字节。这里定义了一个长度为 1024 字节的结构，并将其命名为 kilo。该结构包含一个 1020 字节的数组和一个 4 字节的指针。

函数 FreeMen()使用了第 15 天的课程介绍的链表技术。简单地说，链表是由结构组成的，而结构包含一个指向该结构类型的指针（和其他成员）。另外，还有一个头指针（变量 head，是一个 kilo 指针），它指向链表的第一个节点；而第一个节点指向第二个，第二个指向第三个，依此类推。链表的最后一个节点包含的指针为 NULL。更详细的信息，请参阅第 15 天的课程。

20.2.2　使用 calloc()函数分配内存

calloc()函数也用于分配内存。与 malloc()分配一组字节不同，calloc()分配一组对象。该函数的原型如下：

```
void *calloc(size_t num, size_t size);
```

前面介绍过，在大多数编译器中，size_t 是 unsigned 的别名。参数 num 是要分配的对象数，而 size 是每个对象的大小（单位为字节）。如果分配成功，则将所有内存中的内容清除（设置为 0），并返回指向第一个字节的指针；如果分配失败或者 num 或 size 的值为 0，则返回一个 NULL 指针。

程序清单 20.3 演示了 calloc()函数的用法。

程序清单 20.3　　calloc.c：使用 calloc()动态地分配内存

```
1:  /* Demonstrates calloc(). */
2:
3:  #include <stdlib.h>
4:  #include <stdio.h>
5:
6:  int main( void )
7:  {
8:     unsigned long num;
9:     int *ptr;
10:
11:    printf("Enter the number of type int to allocate: ");
12:    scanf("%ld", &num);
```

www.plcworld.cn


```
13:
14:     ptr = (int*)calloc(num, sizeof(long long));
15:
16:     if (ptr != NULL)
17:        puts("Memory allocation was successful.");
18:     else
19:        puts("Memory allocation failed.");
20:     return 0;
21: }
```

该程序的运行情况如下：

```
Enter the number of type int to allocate: 100
Memory allocation was successful.
Enter the number of type int to allocate: 99999999
Memory allocation failed.
```

该程序第 11 和 12 行提示用户输入一个值，并读取它。这个值决定了程序将试图分配多少内存。第 14 行试图分配足够的内存来存储指定数目的 long long 变量。如果分配失败，calloc()将返回一个 NULL 指针；否则返回一个指向分配的内存的指针。在这个程序中，calloc()返回的值被赋给 int 指针 ptr。第 16～19 行的 if 语句根据 ptr 的值来确定分配成功还是失败，并打印相应的消息。

请输入不同的值，看看能够成功地分配多少内存。最大的量取决于系统的配置。有些系统可以为 25000 个 long long 变量分配内存，而试图为 30000 个这样的变量分配内存时将失败。long long 变量的长度随系统而异。在安装了几百兆内存的系统上，您可能需要输入一个非常大的值，才能导致内存分配失败。

20.2.3 使用 realloc()函数分配更多的内存

realloc()函数用于改变使用 malloc()或 calloc()分配的内存块的大小，其原型如下：

```
void *realloc(void *ptr, size_t size);
```

其中 ptr 是指向原来内存块的指针。新的大小由 size 指定（单位为字节）。realloc()的执行结果有多种：

- 如果有足够的空间用于扩大 ptr 指向的内存块，则分配额外的内存，并返回 ptr；
- 如果没有足够的空间用于扩大内存块，则分配一个 size 字节的新内存块，将原来内存块中的数据复制到新内存块的开头。然后释放原来的内存块，并返回指向新内存块的指针；
- 如果 ptr 为 NULL，则类似于 malloc()，分配一个 size 字节的内存块，并返回一个指向该内存块的指针；
- 如果 size 为 0，则释放 ptr 指向的内存，并返回 NULL；
- 如果没有足够的可用内存来完成重新分配（扩大原来的内存块或分配新内存块），则返回 NULL，而原来的内存块保持不变。

程序清单 20.4 演示了 realloc()的用法。

程序清单 20.4　　realloc.c：使用 realloc()扩大动态分配的内存块

```
1:  /* Using realloc() to change memory allocation. */
2:
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:  #include <string.h>
6:
7:  int main( void )
8:  {
9:     char buf[80], *message;
10:
```

www.plcworld.cn


```
11:    /* Input a string. */
12:
13:    puts("Enter a line of text.");
14:    gets(buf);
15:
16:    /* Allocate the initial block and copy the string to it. */
17:
18:    message = realloc(NULL, strlen(buf)+1);
19:    strcpy(message, buf);
20:
21:    /* Display the message. */
22:
23:    puts(message);
24:
25:    /* Get another string from the user. */
26:
27:    puts("Enter another line of text.");
28:    gets(buf);
29:
30:    /* Increase the allocation, then concatenate the string to it. */
31:
32:    message = realloc(message,(strlen(message) + strlen(buf)+1));
33:    strcat(message, buf);
34:
35:    /* Display the new message. */
36:    puts(message);
37:    return 0;
38: }
```

该程序的运行情况如下：

```
Enter a line of text.
This is the first line of text.
This is the first line of text.
Enter another line of text.
This is the second line of text.
This is the first line of text.This is the second line of text.
```

分析：该程序的第 14 行读取一个字符串，并将其存储到字符数组 buf 中。然后，第 19 行将该字符串复制到 message 指向的内存单元中。第 18 行调用 realloc()来为 message 分配空间，虽然以前并没有为其分配内存。通过将第一个参数设置为 NULL，让 realloc()知道这是首次为 message 分配空间。

第 28 行读取另一个字符串，并将其存储到 buf 中。然后将该字符串拼接到 message 指向的字符串中。由于 message 指向的内存空间只够存储第一个字符串，因此需要重新分配足够的内存来存储第一个和第二个字符串，这是在第 32 行完成的。最后，程序打印拼接后的字符串。

20.2.4　使用 free()函数释放内存

使用 malloc()或 calloc()分配内存时，分配的是动态内存池中可用的内存。动态内存池有时也被称为堆（heap），它可提供的内存是有限的。程序使用完动态分配的内存块后，应将其释放，以便以后可以使用它。要释放动态分配的内存，可使用 free()函数，该函数的原型如下：

```
void free(void *ptr);
```

free()函数释放 ptr 指向的内存，这些内存必须是使用 malloc()、calloc()或 realloc()分配的。如果 ptr 为 NULL，则 free()什么也不做。程序清单 20.5 演示了如何使用 free()函数（程序清单 20.2 也使用了该函数）。

www.plcworld.cn

程序清单20.5　　free.c：使用free()释放动态分配的内存

```
1:  /* Using free() to release allocated dynamic memory. */
2:
3:  #include <stdio.h>
4:  #include <stdlib.h>
5:  #include <string.h>
6:
7:  #define BLOCKSIZE 3000000
8:
9:  int main( void )
10: {
11:     void *ptr1, *ptr2;
12:
13:     /* Allocate one block. */
14:
15:     ptr1 = malloc(BLOCKSIZE);
16:
17:     if (ptr1 != NULL)
18:        printf("\nFirst allocation of %d bytes successful.",BLOCKSIZE);
19:     else
20:     {
21:        printf("\nAttempt to allocate %d bytes failed.\n",BLOCKSIZE);
22:        exit(1);
23:     }
24:
25:     /* Try to allocate another block. */
26:
27:     ptr2 = malloc(BLOCKSIZE);
28:
29:     if (ptr2 != NULL)
30:     {
31:        /* If allocation successful, print message and exit. */
32:
33:        printf("\nSecond allocation of %d bytes successful.\n",
34:              BLOCKSIZE);
35:        exit(0);
36:     }
37:
38:     /* If not successful, free the first block and try again.*/
39:
40:     printf("\nSecond attempt to allocate %d bytes failed.",BLOCKSIZE);
41:     free(ptr1);
42:     printf("\nFreeing first block.");
43:
44:     ptr2 = malloc(BLOCKSIZE);
45:
46:     if (ptr2 != NULL)
47:        printf("\nAfter free(), allocation of %d bytes successful.\n",
48:              BLOCKSIZE);
49:     return 0;
50: }
```

www.plcworld.cn

该程序的输出如下：

```
First allocation of 3000000 bytes successful.
Second allocation of 3000000 bytes successful.
```

分析：该程序尝试动态地分配两个内存块。它使用常量 BLOCKSIZE 来决定分配多少内存。第 15 行使用 malloc()进行第一次分配。第 17～23 行检查返回的值是否为 NULL，以确定分配成功还是失败。然后显示一条消息，指出是否分配成功。如果分配失败，则程序退出。第 27 行尝试分配第二个内存块，同样也检查分配是否成功（第 29～36 行）。如果成功，则调用 exit()来结束程序；如果失败，则打印一条消息，指出试图分配内存时失败。然后使用 free()释放第一个内存块（第 41 行），并再次尝试分配第二个内存块。

您可能需要修改符号常量 BLOCKSIZE 的值。在有些系统上，当该符号常量的值为 3000000 时，程序的输出将如下：

```
First allocation of 3000000 bytes successful.
Second attempt to allocate 3000000 bytes failed.
Freeing first block.
After free(), allocation of 300000 bytes successful.
```

当然，在使用虚拟内存的系统上，分配几乎总是成功。

应　该	不 应 该
使用完动态分配的内存后，一定要将它释放。	不要假设调用 malloc()、calloc()和 realloc()时，总会成功；换句话说，一定要检查是否成功地分配了内存。

20.3　操纵内存块

至此，今天介绍了如何分配和释放内存块。C 语言库中还包含了可用于操纵内存块（将内存块中的所有字节设置为指定的值，将一个内存单元中的信息复制和移动到另一个单元中）的函数。

20.3.1　使用 memset()函数初始化内存

要将内存块中的所有字节设置为特定的值，可以使用 memset()函数。该函数的原型如下：

```
void *memset(void *dest, int c, size_t count);
```

其中参数 dest 指向要初始化的内存块；c 是要设置的值；count 指定要从 dest 开始设置多少个字节。注意，虽然 c 的类型为 int，但它被视为 char。换句话说，只有低端字节被使用，因此只可以将它设置为 0～255 的值。

memset()用于将内存块设置为指定的值。由于该函数只能使用 char 类型的初始值，因此对于操纵数据类型不为 char 的内存块而言，它没有多大的用处，除非您要将内存块的值初始化为 0。换句话说，如果要将 int 数组的所有元素都初始化为 99，则使用 memset()的效率不高，但可以使用它来将所有的元素初始化为 0。memset()函数的用法将在程序清单 20.6 中进行演示。

20.3.2　使用 memcpy()复制内存中的数据

memcpy()用于在内存块（有时候也被称为缓冲区）之间复制数据。该函数不关心被复制的数据的类型——而只是逐字节地进行复制，它的原型如下：

```
void *memcpy(void *dest, void *src, size_t count);
```

其中参数 dest 和 src 分别指向目标内存块和源内存块；count 指定要复制多少个字节。函数的返回值为 dest。如果两个内存块重叠在一起，则该函数可能无法正常运行——src 中的有些数据被复制之前，可能被覆盖了。对于内存块相互重叠的情况，可以使用下一节将介绍的 memmove()函数来处理。函数 memcpy()的用法将在程序清单 20.6 中进行演示。

20.3.3　使用 memmove()函数移动内存中的数据

memmover()与 memcpy()类似，也是将一个内存块中指定数目的字节复制到另一个内存块中。然而，它

www.plcworld.cn

更为灵活，因为它能够处理相互重叠的内存块。由于 memcpy()具备的功能 memmove()也有，但在处理相互重叠的内存块时，后者更灵活，因此没有理由使用 memcpy()。memmove()的原型如下：

```
void *memmove(void *dest, void *src, size_t count);
```

其中，参数 dest 和 src 分别指向目标内存块和源内存块，count 指定要复制多少个字节。函数的返回值为 dest。如果内存块相互重叠，该函数能确保源内存块的重叠区域中的数据被覆盖之前得以复制。程序清单 20.6 演示了函数 memset()、memcpy()和 memmove()的用法。

程序清单 20.6　　mem.c：演示函数 memset()、memcpy()和 memmove()的用法

```
1:  /* Demonstrating memset(), memcpy(), and memmove(). */
2:
3:  #include <stdio.h>
4:  #include <string.h>
4:
5:  char message1[60] = "Four score and seven years ago ...";
6:  char message2[60] = "abcdefghijklmnopqrstuvwxyz";
7:  char temp[60];
8:
9:  int main( void )
10: {
11:    printf("\nmessage1[] before memset():\t%s", message1);
12:    memset(message1 + 5, '@', 10);
13:    printf("\nmessage1[] after memset():\t%s", message1);
14:
15:    strcpy(temp, message2);
16:    printf("\n\nOriginal message: %s", temp);
17:    memcpy(temp + 4, temp + 16, 10);
18:    printf("\nAfter memcpy() without overlap:\t%s", temp);
19:    strcpy(temp, message2);
20:    memcpy(temp + 6, temp + 4, 10);
21:    printf("\nAfter memcpy() with overlap:\t%s", temp);
22:
23:    strcpy(temp, message2);
24:    printf("\n\nOriginal message: %s", temp);
25:    memmove(temp + 4, temp + 16, 10);
26:    printf("\nAfter memmove() without overlap:\t%s", temp);
27:    strcpy(temp, message2);
28:    memmove(temp + 6, temp + 4, 10);
29:    printf("\nAfter memmove() with overlap:\t%s\n", temp);
30:    return 0;
31: }
```

该程序的输出如下：

```
message1[] before memset():     Four score and seven years ago ...
message1[] after memset():      Four @@@@@@@@@@seven years ago ...

Original message: abcdefghijklmnopqrstuvwxyz
After memcpy() without overlap: abcdqrstuvwxyzopqrstuvwxyz
After memcpy() with overlap:    abcdefefefefefefqrstuvwxyz

Original message: abcdefghijklmnopqrstuvwxyz
After memmove() without overlap:        abcdqrstuvwxyzopqrstuvwxyz
```

www.plcworld.cn


```
After memmove() with overlap:   abcdefefghijklmnqrstuvwxyz
```

分析：memset()的工作原理很简单。请注意程序使用指针表示法 message1 + 5 来指定 memset()从 message1[] 的第 6 个字符（别忘了，数组的索引从 0 开始）开始设置。因此，messag1[]的第 6～15 个字符被修改为@。

当源内存块和目标内存块不相互重叠时，memcpy()能正常地工作。temp[]的 10 个字符（从位置 17 开始，即字母 q～z）被复制到位置 5～14 中（原来存储的是 e～n）。然而，如果源内存块和目标内存块相互重叠，则情况将不同。当该函数试图将从位置 4 开始的 10 个字符复制到从位置 6 开始的内存中时，有 8 个位置是相互重叠的。您可能认为，字母 e-n 将被复制，并覆盖原来的字母 g～p。但是情况并非如此，字母 e 和 f 将重复 5 次。

如果内存块没有相互重叠，则 memove()的功能与 memcpy()相同；但如果有重叠，memmove()将把源内存块中的字符复制到目标内存块中。

应　该	不 应 该
如果内存块相互重叠，则一定要使用 memmove()，而不是 memcpy()	不要试图使用 memset()将 int、float 或 double 数组初始化为非零值。

20.4　位的用法

您可能知道，计算机存储空间的最小单位为位。有时候，如果能操纵数据中的位，将很有帮助。C 语言提供了多种用于操纵位的工具。

C 语言的按位运算符使您能够操纵整型变量中的各个位。别忘了，位是数据存储空间的最小单位，其取值只有两个：0 或 1。按位运算符只能用于整型变量：char、int 和 long。阅读接下来的内容之前，您应该熟悉二进制表示法——计算机内部存储整数的方式。如果您需要复习二进制表示法，请参阅附录 C。

当程序需要直接与计算机硬件交互（这一主题超出了本书的范围）时，常常要使用按位运算符。按位运算符也有其他的用途，下面将介绍这些用途。

20.4.1　移位运算符

有两种移位运算符，它们将整型变量中的位移动指定数目的位置。运算符<<将位左移，而运算符>>右移。这两个双目运算符的语法如下：

```
x << n
x >> n
```

每个运算符都将 x 中的位沿指定的方向移动 n 个位置。右移时，将在变量最左边的 n 位中加上 0；而左移时，将在最右边的 n 位中加上 0。下面是几个例子：

- 将二进制数 00001100（十进制数 12）向右移两位时，结果为 00000011（十进制数 3）；
- 将 00001100 向左移三位时，结果为 01100000（十进制数 96）；
- 将 00001100 向右移三位时，结果为 00000001（十进制数 1）；
- 将 00110000（十进制数 48）向左移 3 位时，结果为 10000000（十进制数 128）。

在有些情况下，可以使用移位运算符将整型变量乘以和除以 2^n。将整数左移 n 位相当于将其乘以 2^n；而右移 n 位相当于除以 2^n。仅当左移不会导致溢出（没有位因为被移出最左边而丢失）时，这种通过左移来实现乘法的结果才准确。通过右移来实现除法时，结果中的小数部分将丢失。例如，如果将 5（二进制数 00000101）右移一位，以便将其除以 2，则结果将为 2（二进制数 00000010），而不是 2.5，因为小数部分（.5）丢失了。程序清单 20.7 演示了移位运算符的用法。

程序清单 20.7　　**shiftit.c：使用移位运算符**

```
1:  /* Demonstrating the shift operators. */
2:
3:  #include <stdio.h>
```

www.plcworld.cn


```
4:
5: int main( void )
6: {
7:      unsigned int y, x = 255;
8:      int count;
9:
10:     printf("Decimal\t\tshift left by\tresult\n");
11:
12:     for (count = 1; count < 8; count++)
13:     {
14:         y = x << count;
15:         printf("%d\t\t%d\t\t%d\n", x, count, y);
16:     }
17:     printf("\n\rDecimal\t\tshift right by\tresult\n");
18:
19:     for (count = 1; count < 8; count++)
20:     {
21:         y = x >> count;
22:         printf("%d\t\t%d\t\t%d\n", x, count, y);
23:     }
24:     return 0;
25: }
```

该程序的输出如下：

```
Decimal         shift left by   result
255             1               254
255             2               252
255             3               248
255             4               240
255             5               224
255             6               192
255             7               128
Decimal         shift right by  result
255             1               127
255             2               63
255             3               31
255             4               15
255             5               7
255             6               3
255             7               1
```

20.4.2 按位逻辑运算符

有三种按位逻辑运算符可用于操纵整型数据中的位，如表 20.1 所示。这些运算符的名称与以前介绍的TRUE/FALSE 逻辑运算符类似，但工作原理完全不同。

表 20.1　　按位逻辑运算符

运 算 符	操　作
&	与（AND）
\|	或（OR）
^	异或

www.plcworld.cn

这些运算符都是双目的，它们根据两个操作数的相应位的值，将结果的位设置为 0 或 1。它们的工作原理如下：

- 仅当两个操作数的相应位皆为 1 时，按位 AND 才将结果位设置为 1；否则设置为 0。AND 运算符用于关闭（清除）值中的一个或多个位；
- 仅当两个操作数的相应位皆为 0 时，按位或才将结果位设置为 0；否则设置为 1。OR 运算符用于开启（设置）值中的一个或多个位；
- 仅当两个操作数的相应位不同时（一个为 0，一个为 1），按位异或才将结果位设置为 1；否则设置为 0。

表 20.2 的范例说明了这些运算符的工作原理。

表 20.2　　按位逻辑运算符的工作原理

运　算	范　例
AND	11110000 & 01010101 -------- 01010000
OR	11110000 \| 01010101 -------- 11110101
异或	11110000 ^ 01010101 -------- 10100101

前面指出过，按位运算 AND 和 OR 可用于清除和设置整型值的指定位。下面解释这句话的含义。假设有一个 char 变量，您想清除它的第 0 和 4 位（将其设置为 0），其他位保持不变。为此，可将该变量与另一个值（11101110）进行按位 AND 运算。原理如下：

在第二个值中为 1 的每个位置，结果将与原来变量的相应位置相同：

```
0 & 1 == 0
1 & 1 == 1
```

在第二个值中为 0 的每个位置，结果将为 0，而不管原来变量的相应位置是什么：

```
0 & 0 == 0
1 & 0 == 0
```

使用 OR 运算来设置位的原理与此类似。在第二个值中为 1 的每个位置，结果为 1；而在第二个值中为 0 的每个位置，结果将与原来变量的相应位置相同：

```
0 | 1 == 1
1 | 1 == 1
0 | 0 == 0
1 | 0 == 1
```

20.4.3　求补运算符

这里要介绍的最后一个按位运算符是求补运算符（~），它是一个单目运算符。它将操作数的每一位反转，即将所有的 0 变成 1，将所有的 1 改为 0。例如~254（二进制数 11111110）等于 1（二进制数 00000001）。

20.4.4　结构中的位字段

最后一个与位相关的主题是结构中的位字段。第 11 天的课程介绍了如何根据程序的需求，定义自己的结

www.plcworld.cn

构。通过使用位字段，可以实现更大程度的定制，并节省内存空间。

位字段（bit field）是一个结构成员，包含指定数目的位。可以声明一个一位、两位或任何数目位的位字段。这样做有什么好处呢？

假设您要编写一个雇员数据库程序，用于记录公司的雇员。数据库存储的信息中，很多数据项为 yes/no，如雇员是否是大学毕业。这种 yes/no 信息都可以用一位来存储，其中 1 表示是，0 表示否。

在 C 语言的标准数据类型中，最小的类型为 char。确实可以使用 char 类型的结构成员来存储 yes/no 数据，但 char 变量中的七位被浪费掉了。通过使用位字段，可以在一个 char 变量中存储 8 个 yes/no 值。

位字段不仅仅可用于存储 yes/no 值。还是以前面的数据库为例，假设公司有三个不同的健康保险计划，而数据库需要记录每个雇员参加的保险计划。您可以使用 0 来表示没有参加任何保险计划，1、2、3 分别表示参加了第 1、2、3 种保险计划。为此，使用一个两位的位字段便足够了，因为两位可以表示 0～3。同样，三位可以表示 0～7，四位可以表示 0～15，等等。

位字段的命名和存取方式与常规的结构成员相同。所有位字段的类型都为 unsigned int，并可在成员名后加上冒号和数字来指定其长度（单位为位）。下面的代码声明了一个结构，该结构包含一个名为 dental 的一位成员、一个名为 college 的一位成员和一个名为 health 的两位成员：

```
struct emp_data
{
    unsigned dental     : 1;
    unsigned college    : 1;
    unsigned health     : 2;
    ...
};
```

省略号表示其他结构成员。成员可以是位字段，也可以是常规数据类型。注意，位字段必须放在结构定义的最前面，然后才能是其他非位字段成员。要存取位字段，可使用成员结构运算符，就像存取其他结构成员一样。例如，可以对上述结构进行扩展，使之更有用：

```
struct emp_data
{
    unsigned dental     : 1;
    unsigned college    : 1;
    unsigned health     : 2;
    char fname[20];
    char lname[20];
    char ssnumber[10];
};
```

然后，可以声明一个结构数组：

```
struct emp_data workers[100];
```

要给第一个数组元素赋值，可以这样编写代码：

```
workers[0].dental = 1;
workers[0].college = 0;
workers[0].health = 2;
strcpy(workers[0].fname, "Mildred");
```

当然，使用一位的字段时，如果采用符号常量 YES 和 NO（它们的值为 1 和 0），则代码将更为清晰。您将位字段看作一个小型的、包含指定位数的无符号整数。对于包含 n 位的位字段，可以将 0～2^{n-1} 的值赋给它。如果您将超出范围的值赋给位字段，编译器不会报错，但结果将是不可预测的。

应　读	不 应 读
使用位字段时，一定要应用常量 YES 和 NO 或者 TRUE 和 FALSE，这样，与使用 0 和 1 相比，代码的可读性将更高。	不要定义一个包含 8 或 16 位的位字段，因为可以使用 char 或 int 变量来实现这种字段。

www.plcworld.cn

20.5　总　结

今天的课程介绍了很多 C 语言编程方面的主题，包括如何分配、重新分配和释放内存，同时介绍了一些使您能够灵活地为数据分配空间的命令。还介绍了如何和何时对变量和指针执行强制类型转换。忘记或错误使用强制类型转换是导致难以查找的程序 bug 的常见原因，因此有必要复习这一主题。另外，还介绍了如何使用 memset()、memcpy()和 memmove()函数来操纵内存块。最后，介绍了如何在程序中操纵和使用位。

20.6　问与答

问：动态内存分配有何优势？只需在源代码中声明所需的存储空间不就可以了吗？

答：如果在源代码中声明所有的数据存储空间，则程序可用的内存量将是固定的。在编写程序时，您必须预先知道需要多少内存。动态内存分配使程序能够根据当前的情况和用户输入，控制所需的内存量。程序可以根据需要使用内存，但最多不能超过计算机中的可用内存。

问：为何需要释放内存？

答：您刚开始学习 C 语言时，编写的程序都不大。但随着程序越来越大，它使用的内存也越来越多。编写程序时，应尽可能高效地使用内存。使用完内存后，必须将其释放。如果您编写的是在多任务环境中运行的程序，则其他程序可能需要使用您未使用的内存。虽然在有些系统中，当程序结束时，将自动收回内存，但并非所有的系统都会这样做。

问：如果重用一个字符串，而没有调用 realloc()，情况将如何？

答：如果您使用的字符串有足够的空间，则不必调用 realloc()；仅当当前字符串不够大时，才需要调用 realloc()。别忘了，C 编译器让您能够做任何事，即使是不应该做的事。可以将一个字符串存储到另一个字符串中，只要前者有足够的空间。然而，如果没有足够的空间，则后面的内存也将被覆盖。这些内存中可能没有任何数据，也可能存储了非常重要的数据。如果需要分配更多的内存，可以调用 realloc()。

问：函数 memset()、memcpy()和 memmove()有何优势？使用循环，通过赋值语句来初始化或复制内存不就可以了吗？

答：在有些情况下，可以使用循环，通过赋值语句来初始化内存。事实上，有时候只能采用这种方式，例如将 float 数组的所有元素设置为 1.23。而在其他情况下，内存没有被分配给数组或链表，因此只能使用 mem...()函数。另外，在有些情况下，使用循环和赋值语句是管用的，但使用 mem...()函数更简单，速度也更快。

问：在什么情况下需要使用移位运算符和按位逻辑运算符？

答：这些运算符最常用于直接同计算机硬件交互——通常需要生成和解释特定的位模式。这一主题超出了本书的范围。即使您永远不需要直接操纵硬件，在某些情况下，也可以使用移位运算符将一个整型值乘以或除以 2^n。

问：使用位字段确实有那么多的好处吗？

答：是的。使用位字段的好处很多。请考虑与本章的范例类似的情况：一个文件中存储了调查信息。被调查者对提出的问题做出 TRUE 或 FALSE 回答。如果向 1 万个被调查者询问了 100 个问题，并使用 char 变量来存储答案，则需要 10000×100（100 万）个字节的内存（char 变量的长度为 1 字节）。如果使用位字段，为每个答案分配一位，则只需 10000×100 位的内存。由于一个字节为 8 位，因此这相当于 130000 字节，比 100 万要少得多。

www.plcworld.cn

20.7 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

20.7.1 小测验

1. 内存分配函数 malloc()和 calloc()之间有何区别？
2. 对数值变量进行强制类型转换的最常见的原因是什么？
3. 下列各个表达式的类型是什么？假设 c 的类型为 char，i 的类型为 int，l 的类型为 long，f 的类型为 float。

 a. `( c + i + l )`

 b. `( i + 32 )`

 c. `( c + 'A' )`

 d. `( i + 32.0 )`

 e. `( 100 + 1.0 )`

4. 动态地分配内存意味着什么？
5. 函数 memcpy()和 memmove()之间有何区别？
6. 假设您的程序使用了一个结构，该结构的一个成员将星期几存储为 1～7 的值，则如何做才能使内存的使用效率最高？
7. 要存储当前的日期，最少需要多少内存（提示：月/日/年，将年份视为离 1900 年多久）？
8. 10010010 << 4 的结果为多少？
9. 10010010 >> 4 的结果是多少？
10. 描述下述两个表达式的结果之间的差异。

```
(01010101 ^ 11111111 )
( ~01010101 )
```

20.7.2 练习

1. 编写一行代码，使用 malloc()函数分配用于存储 1000 个 long 的内存。
2. 编写一行代码，使用 calloc()函数分配用于存储 1000 个 long 的内存。
3. 假设声明了下述数组：

```
float data[1000];
```

采用两种方式来将该数组的每个元素初始化为 0。其中一种方式使用循环和赋值语句；另一种方式使用 memset()函数。

4. 排错：下述代码有错误吗？

```
void func()
{
    int number1 = 100, number2 = 3;
    float answer;
    answer = number1 / number2;
    printf("%d/%d = %lf", number1, number2, answer)
}
```

5. 排错：下述代码有何错误？

```
void *p;
p = (float*) malloc(sizeof(float));
*p = 1.23;
```

www.plcworld.cn

6. 排错：可以定义下面这样的结构吗？

```
struct quiz_answers
{
    char student_name[15];
    unsigned answer1   : 1;
    unsigned answer2   : 1;
    unsigned answer3   : 1;
    unsigned answer4   : 1;
    unsigned answer5   : 1;
}
```

附录 F 没有提供下述练习的答案。

7. 编写一个使用所有按位逻辑运算符的程序。这些运算符首先被用于一个数字，然后再用于结果。您应该查看程序的输出，以确保您理解了发生的情况。

8. 编写一个程序，它显示一个数字对应的二进制值。例如，如果用户输入 3，则程序应该显示 00000011（提示：需要使用按位运算符）。

www.plcworld.cn

第 21 天课程　编译器的高级用法

这是 21 天学通 C 语言的最后一天的课程。至此，您几乎已经学习了有关 C 语言编程的所有重要主题。今天将介绍 C 语言的其他一些特性，包括：

- 使用多个源代码文件；
- 使用预处理器；
- 使用命令行参数。

21.1　使用多个源代码文件的编程

到目前为止，您编写的所有 C 程序都只包含一个源代码文件（当然，头文件除外）。通常，只需要一个源代码文件，尤其是对于小型程序而言；但是，也可以将程序的源代码分为两个或更多的文件，这被称为模块化编程。为何要这样做呢？接下来的几节将对此进行解释。

21.1.1　模块化编程的优点

使用模块化编程的主要原因与结构化编程及其对函数的依赖紧密相关。随着您的编程经验越来越丰富，您开发的函数将越来越通用，它们不但可用于最初的程序中，还可用于其他程序。例如，您可能编写一组用于将信息显示到屏幕上的通用函数。通过将它们存储在一个独立的文件中，您可以在其他需要将信息显示到屏幕的程序中再次使用它们。当程序包含多个源代码文件时，其中的每个文件被称为模块。

21.1.2　模块化编程技术

每个C 程序只有一个 main()函数。包含 main()函数的模块被称为主模块，其他的模块被称为辅助模块（secondary module）。通常每个辅助模块被存储在一个独立的文件中（后面将解释这样做的原因）。现在来看一些例子，它们说明了多模块编程的基本知识。程序清单 21.1～21.3 分别是一个程序的主模块、辅助模块和头文件，该程序读取用户输入的数字，并显示其平方。

程序清单 21.1　　list2101.c：主模块

```
1:  /* Inputs a number and displays its square. */
2:
3:  #include <stdio.h>
4:  #include "calc.h"
5:
6:  int main( void )
7:  {
8:     int x;
9:
10:    printf("Enter an integer value: ");
```

www.plcworld.cn


```
11:    scanf("%d", &x);
12:    printf("\nThe square of %d is %ld.\n", x, sqr(x));
13:    return 0;
14: }
```

程序清单 21.2　　calc.c：辅助模块

```
1:  /* Module containing calculation functions. */
2:
3:  #include "calc.h"
4:
5:  long sqr(int x)
6:  {
7:     return ((long)x * x);
8:  }
```

程序清单 21.3　　calc.h：calc.c 包含的头文件

```
1:  /* calc.h: header file for calc.c. */
2:
3:  long sqr(int x);
4:
5:  /* end of calc.h */
```

该程序的运行情况如下：

```
Enter an integer value: 100

The square of 100 is 10000.
```

分析：下面详细介绍这三个文件的组成部分。头文件 calc.h 包含 calc.c 使用的函数 sqr()的原型。由于使用 sqr()的模块都需要知道 sqr()的原型，因此必须将它放在 calc.h 中。

辅助模块文件 calc.c 包含了 sqr()函数的定义。编译指令#define 被用来包含头文件 calc.h。注意，使用双引号而不是尖括号将头文件名括起（后面将解释这样做的原因）。

主模块 list2101.c 包含 main()函数。该模块还包含头文件 calc.h。

使用编辑器创建上述三个文件后，如何进行编译和链接，以生成可执行文件呢？编译器会替您进行控制。请在命令行输入下述命令：

```
xxx list2101.c calc.c
```

其中 xxx 是用于执行编译器的命令。上述命令让编译器完成以下任务：

1. 编译 list2101.c，创建 list2101.obj（在 UNIX 系统中，为 list2101.o）。如果有错误，编译器显示错误消息。

2. 编译 calc.c，创建 calc.obj（在 UNIX 系统中，为 calc.o）。同样，如果有错误，则显示错误消息。

3. 链接 list2101.obj、calc.obj 和所需的标准库函数，以生成可执行文件 list2101.exe。

如果您使用的是集成开发环境，则可以创建一个工程，让它来负责编译多个源代码文件。例如，如果您使用的是 Dev-C++编译器，则可以按下面的步骤来编译前面的程序清单：

1. 打开 Dev-C++；

2. 选择菜单 File/New Project，打开如图 21.1 所示的对话框。

3. 选中单选按钮 C project，然后单击 Console Application，打开如图 21.2 所示的 New Project Name 对话框。

4. 输入工程名，如 List2101，并保存该工程。

www.plcworld.cn

5. IDE 将打开一个未命名的 C 语言源代码文件，如图 21.3 所示。然后您便可以输入代码或将文件关闭。如果输入了代码，则应保存该文件。然后选择菜单 Project/Add Source File，并输入其他两个源文件的代码。

6. 如果源文件已经创建好，则可以将它们加入到新建的工程中。方法是，选择 Project/Add to project 菜单，然后选择要加入的文件。这样，这些文件将被加入到 IDE 中。

7. 按以前介绍的方法编译该工程。

有关在 Dev-c++中编译工程的更详细的信息，请参阅帮助文档。

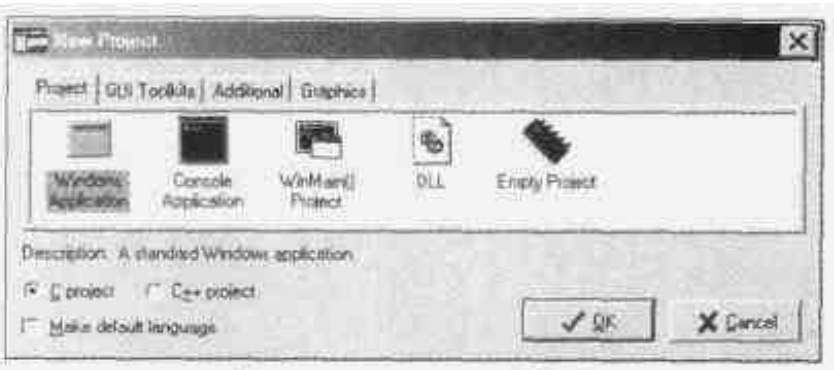


图 21.1　New Project 对话框

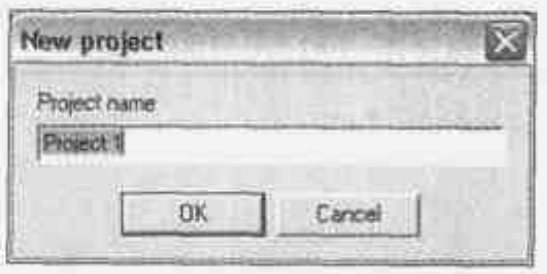


图 21.2　New Project Name 对话框

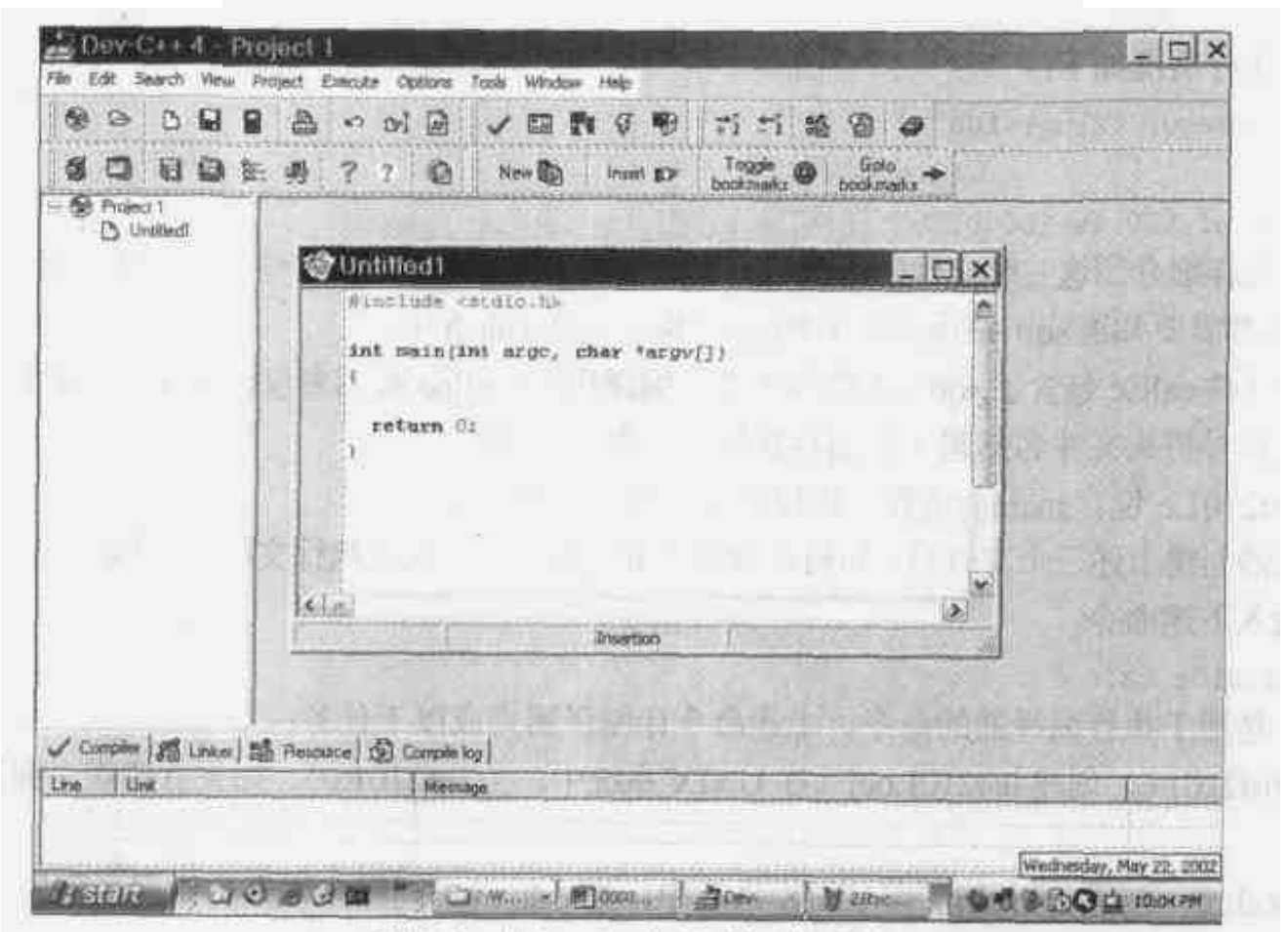


图 21.3　在 Dev-C++中打开的一个工程

21.1.3　模块的组成部分

正如您看到的，编译和链接多模块程序的机制很简单。唯一的问题是每个文件中应包含哪些内容。本节将介绍一些通用的指导原则。

辅助模块应包含通用的函数，即要在其他程序中使用的函数。常见的做法是，对于每一类函数，创建一个辅助模块。例如，将键盘函数放在 keyboard.c 中，屏幕显示函数放在 screen.c 中，等等。要编译并链接两

www.plcworld.cn

个以上的模块，只需在命令行中列出所有的源代码文件：

```
tcc mainmod.c screen.c keyboard.c
```

主模块应包含 main()和其他程序特有（即不通用）的函数。

通常，每个辅助模块都有一个头文件。头文件的名称与相应的模块名相同，但扩展名为.h。头文件应包含以下内容：

- 辅助模块中的函数的原型；
- 定义模块使用的宏和符号常量的#define 编译指令；
- 模块使用的结构和外部变量的定义。

由于头文件可能被多个源代码文件包含，因此要防止其某些部分被编译多次。为此，可以使用预处理器来实现有条件的编译（这将在本章后面介绍）。

21.1.4　外部变量和模块化编程

在很多情况下，主模块和辅助模块之间的数据通信只是通过给函数传递参数以及函数返回值来进行的。在这种情况下，无需关心数据的可见性；但如果希望一个外部变量在主模块和辅助模块中都可见，该如何办呢？

第 12 天的课程中介绍过，外部变量是在函数外面声明的变量。外部变量在其所属的整个源代码文件中都是可见的，但在其他模块中，并不会自动地可见。要使之可见，必须在每个模块中使用关键字 extern 来声明它。例如，如果在主模块中声明了一个这样的外部变量：

```
float interest_rate;
```

则可以在辅助模块中进行如下声明（在函数外面），使之在该模块中是可见的：

```
extern float interest_rate;
```

关键字 extern 告诉编译器，变量 interest_rate 的首次声明（预留存储空间的声明）位于其他地方，但应使该变量在本模块中可见。所有的 extern 变量都是静态的，它们在模块中的所有函数中都是可见的。图 21.4 演示了如何在多模块程序中使用关键字 extern。



警告：使用 extern 来声明外部变量，而又没有在其他地方实际声明该变量时，将发生错误。这种错误可能在链接时发生，也可能在运行时发生。

在图 21.4 中，变量 x 在所有三个模块中都是可见的；而变量 y 只在主模块和辅助模块 1 中是可见的。

```
/* main module */
int x, y;
main()
{
...
...
}
```

```
/* secondary mod1.c */
extern int x, y;
func1()
{
...
}
...
```

```
/* secondary mod2.c */
extern int x;
func4()
{
...
}
...
```

图 21.4　使用关键字 extern 使外部变量在整个模块中可见

21.1.5　使用.obj 文件

编写并仔细地调试辅助模块后，在其他程序中使用它时，无需重新进行编译。有了模块代码的目标文件

www.plcworld.cn

后，只需将其链接到使用它的程序中即可。

编译程序时，对于其中的每个源代码文件，编译器都将创建一个目标文件，该文件的名称与源代码文件相同，但扩展名为.obj（在 UNIX 系统中，为.o）。例如，假设您开发了一个名为 keyboard.c 的模块，并要将其与主模块 database.c 一起编译，可以使用下面的命令：

```
tcc database.c keyboard.c
```

这样，将在磁盘上创建一个名为 keyboard.obj 的文件。当您确定 keyboard.c 中的函数能够正确运行后，以后重新编译 database.c（或其他需要使用该模块的程序）时，无需再次编译 keyboard.c，而只需链接已有的目标文件（keyboard.obj）。为此，可以使用下面的命令：

```
tcc database.c keyboard.obj
```

这样，编译器将编译 database.c，并将生成的目标文件 database.obj 与 keyboard.obj 链接起来，创建出可执行文件 database.exe。这样可以节省时间，因为编译器不用重新编译 keyboard.c 中的源代码。然而，如果您修改了 keyboard.c 中的代码，则必须重新编译它。另外，如果您修改了头文件，也必须重新编译使用了该头文件的所有模块。

应　该	不应该
对于通用函数，应将其放在单独的源代码文件中。这样，便可以将其链接到其他需要使用它们的文件中。	编译多个源代码文件时，其中只能有一个文件包含 main()函数（任何函数都如此）。 编译多个文件时，不要总是使用源代码文件。如果源代码文件已经被编译为目标文件，则仅当修改了该文件时，才重新编译它。这样可以节省大量的时间。

21.1.6　使用生成工具

几乎所有的C编译器都自带了一个生成工具（make utility），它可以简化并加快处理多个源代码文件的工作。这个工具通常名为 nmake.exe，它使您能够编写生成文件（make file）。之所以称作生成文件是因为它定义了程序各个部分之间的依存性。依存性是什么意思呢？

假设有一个这样的工程，它有一个名为 program.c 的主模块和一个名为 second.c 的辅助模块；还有两个头文件：program.h 和 second.h。program.c 包含了这两个头文件，而 second.c 只包含了 second.h。program.c 中的代码调用了 second.c 中的函数。

program.c 依赖于这两个头文件，因为它包含了它们。如果对其中任何一个头文件进行了修改，则必须重新编译 program.c，使之包含所做的修改。而 second.c 依赖于 second.h，但不依赖于 program.h。如果您修改了 program.h，则无需重新编译 second.c，而只需链接上一次编译 second.c 时创建的目标文件 second.obj 即可。

生成文件描述了工程中的依存性。每当您编写一个或多个源代码文件时，都可以使用 nmake 工具来"启动"生成文件。该工具检查源代码和目标文件的时间和日期邮戳，并根据您定义的依存性，指示编译器只重新编译那些依赖于被修改的文件的文件。这样，避免了不必要的编译，从而使效率最高化。

对于只涉及一两个源代码文件的工程，通常没有必要去定义生成文件。然而，对于大型工程，这样做是有好处的。有关如何使用 nmake 工具，请参阅编译器文档。

21.2　C语言的预处理器

所有的C编译器软件包都提供了预处理器。编译C程序时，程序首先由编译器中的预处理器进行处理。在大多数C编译器中，预处理器都被集成到编译器程序中。当您运行编译器时，它将自动运行预处理器。

预处理器根据源代码中的指令（预处理器编译指令）对源代码进行修改。预处理器输出修改后的源代码文件，然后，该输出被用作下一个编译步骤的输入。通常，您看不到这样的文件，因为编译器使用完毕后，将把它删除。后面将介绍如何查看这种中间文件。这里首先介绍预处理器编译指令，它们都以符号#打头。

www.plcworld.cn

21.2.1　#define 预处理器编译指令

#define 预处理器编译指令有两种用途：创建符号常量和创建宏。

1．使用#define 创建简单的替换宏

第 3 天的课程已经介绍过替换宏，虽然那时使用的术语是符号常量。要创建替换宏，可以使用#define 将一种文本替换为另一种文本。例如，要将 text1 替换为 text2，可以这样编写代码：

```
#define text1 text2
```

上述编译指令导致预编译器将源代码文件中的所有 text1 替换为 text2，但如果 text1 位于双引号中，则不替换。

替换宏最常见的用途是用于创建符号常量，这已经在第 3 天的课程中介绍过了。例如，如果程序中包含下述代码行：

```
#define MAX 1000
x = y * MAX;
z = MAX - 12;
```

则在预处理过程中，上述源代码被修改为：

```
x = y * 1000;
z = 1000 - 12;
```

这相当于使用编辑器的“查找并替换”功能，将所有的 MAX 替换为 1000。当然，原来的源代码文件保持不变，而是创建一个副本，并进行修改。#define 并不局限于创建数值符号常量。例如，您可以编写这样的代码：

```
#define ZINGBOFFLE printf
ZINGBOFFLE("Hello, world.");
```

虽然没有理由这样做。您还需知道，有些作者将#define 定义的符号常量称作宏（符号常量也叫做明显常量（manifest constant））。然而，在本书中，宏指的是下一节将描述的一种结构（construction）。

2．使用#define 创建函数宏

也可以使用编译指令#define 来创建函数宏。函数宏是一种简写，使用简单的东西来表示复杂的东西。其名称中之所以包含“函数”两字，是因为这种宏能接受参数，就像真正的函数那样。函数宏的优点之一是，其参数对类型不敏感。因此，您可以将任何数值类型的变量传递给接受数值参数的函数宏。

请看一个例子。下面的预处理器编译指令：

```
#define HALFOF(value) ((value)/2)
```

定义了一个名为 HALFOF 的宏，该宏接受一个名为 value 的参数。预处理器将源代码中的所有 HALOF 宏替换为相应的定义文本，并在必要时插入参数。因此，下面的代码行：

```
result = HALFOF(10);
```

将被替换为：

```
result = ((10)/2);
```

同样，下面的代码行：

```
printf("%f", HALFOF(x[1] + y[2]));
```

将被替换为：

```
printf("%f", ((x[1] + y[2])/2));
```

宏可以有多个参数，而其中每个参数都可以在替换文本中出现多次。例如，下面的宏计算 5 个数的平均值，它有 5 个参数：

```
#define AVG5(v, w, x, y, z) (((v)+(w)+(x)+(y)+(z))/5)
```

下面的宏使用了每个参数两次（其中的条件运算符确定两个值中哪个更大，条件运算符在第 4 天的课程中介绍过）：

www.plcworld.cn


```
#define LARGER(x, y) ((x) > (y) ? (x) : (y))
```

宏可以有任意数目的参数，但列表中的所有参数都必须出现在替换字符串中。例如，下面的宏定义是非法的，因此参数z没有出现在替换字符串中：

```
#define ADD(x, y, z) ((x) + (y))
```

另外，调用宏时，必须传递正确数目的参数。

编写宏定义时，宏名后必须紧跟左圆括号，中间不能有空白。左圆括号告诉编译器，定义的是一个函数宏，而不是符号常量。请看下面的定义：

```
#define SUM (x, y, z) ((x)+(y)+(z))
```

由于SUM和(之间有空格，因此预处理器将其视为一个简单的替换宏，将源代码中的每个SUM替换为(x, y, z) ((x)+(y)+(z))，这显然不是您希望的。

另外要注意的是，在替换字符串中，每个参数都用圆括号括起。为避免将表达式作为参数传递给宏时产生不希望的副作用，这是必须的。请看下述定义时没有使用圆括号的宏：

```
#define SQUARE(x) x*x
```

如果调用该宏时使用简单变量作为参数，则不会发生问题。但如果您将表达式作为参数，情况将如何呢？

```
result = SQUARE(x + y);
```

得到的宏扩展如下，它计算得到的结果将是不正确的：

```
result = x + y * x + y;
```

使用圆括号可以避免这种问题，如下面的范例所示：

```
#define SQUARE(x) (x)*(x)
```

这样定义被扩展为如下所示，它计算的结果是正确的：

```
result = (x + y)*(x + y);
```

在宏定义中使用字符串化运算符（#，有时也被称为字符串-字面运算符）可以提高灵活性。在替换字符串中，如果宏参数前面有#，则展开宏时，该参数将被转换为用引号括起的字符串。因此，如果定义了一个这样的宏：

```
#define OUT(x) printf(#x)
```

然后使用下面的语句来调用这种宏：

```
OUT(Hello Mom);
```

则上述语句将被展开为：

```
printf("Hello Mom");
```

字符串化运算符执行转换时，会考虑特殊字符。因此，如果参数中的字符需要转义字符，#运算符将在该字符前插入一个反斜杠。继续前面的例子，下述调用：

```
OUT("Hello Mom");
```

将被展开为：

```
printf("\"Hello Mom\"");
```

程序清单21.4演示了#运算符的用法。首先，有必要介绍另一个用于宏中的运算符：拼接运算符（##）。在展开宏时，该运算符将两个字符串拼接起来，但不会加上引号，也不会对转义字符做特殊处理。该运算符的主要用途是创建源代码序列。例如，如果您这样定义并调用一个宏：

```
#define CHOP(x) func ## x
salad = CHOP(3)(q, w);
```

则第二行的宏调用将被展开为：

```
salad = func3 (q, w);
```

从中可以知道，通过使用##运算符，可以决定应调用哪个函数。这实际上修改了源代码。

程序清单21.4是一个使用#运算符的例子。

程序清单21.4　　preproc.c：在宏扩展中使用#运算符

```
1:  /* Demonstrates the # operator in macro expansion. */
2:
3:  #include <stdio.h>
4:
```

www.plcworld.cn


```
5:  #define OUT(x) printf(#x " is equal to %d.\n", x)
6:
7:  int main( void )
8:  {
9:    int value = 123;
10:   OUT(value);
11:   return 0;
12: }
```

该程序的输出如下：

value is equal to 123.

分析：通过在第 5 行使用#运算符，使得扩展宏调用时，变量名 value 被作为用引号括起的字符串传递给 printf()函数。扩展后，第 10 行的宏 OUT 变为：

```
printf("value" " is equal to %d.", value );
```

3. 宏和函数

您知道，可以使用函数宏代替真正的函数，至少在代码相对简短时是这样的。函数宏可以超过一行，但不太可能包含很多行。在可以使用函数，也可以使用宏时，应该如何选择呢？这需要在速度和长度之间进行折衷。

源代码中的所有宏调用都将根据宏定义被扩展为相应的代码。如果程序调用某个宏 100 次，则最后的程序中将包含 100 个扩展宏代码的拷贝；而函数代码只有一个拷贝。因此，就代码长度而言，使用函数更好。

程序调用函数时，转到函数处执行以及从函数返回涉及到一定的处理开销；而调用宏没有任何处理开销，因为宏代码被直接嵌入到程序中。因此，就速度而言，使用函数宏更好。

对于初学者而言，通常并不太关心长度/速度的因素。仅当应用程序非常大，且对时间敏感时，这些因素才重要。

4. 查看宏扩展

有时候，您可能想查看扩展后的宏，尤其是它不能正常工作时。要查看扩展后的宏，可以命令编译器在首次处理代码后，创建一个包含宏扩展的文件。如果 C 编译器使用集成开发环境（IDE），则您可能无法这样做；您必须在命令提示符下运行编译器。大多数编译器都有一个在编译期间应该设置的标记，该标记被作为一个命令行参数传递给编译器。

例如，要使用 Microsoft 的编译器对一个名为 program.c 的程序进行预编译，可以执行下述命令：

```
cl /E program.c
```

对于 UNIX 编译器，可执行下述命令：

```
cc -E program.c
```

预处理器首先对代码进行处理。将所有的头文件包含进来，对#define 宏进行扩展，并执行其他预处理器编译指令。根据使用的编译器，输出可能被发送到 stdout（屏幕），也可能被发送到一个磁盘文件中，该文件的名称与程序名相同，但使用一个特殊的扩展名。Microsoft 的编译器将预处理输出发送到 stdout。不幸的是，将处理后的代码快速地显示到屏幕上毫无用处。可以使用重定向命令将这种输出发送到一个文件中，如下面的范例所示：

```
cl /E program.c > program.pre
```

然后，可以将该文件装入到编辑器中，进行打印或查看。

应　读	不应读
应使用#define 来定义符号常量。符号常量可提高代码的可读性。被定义为符号常量的有颜色、真/假、yes/no、键盘按键和最大值等。本书一直在使用符号常量。	不要滥用函数宏。必要时可以使用，但必须是比使用函数更合适的情况。

www.plcworld.cn

21.2.2 使用编译指令#include

您已经知道如何使用预处理器编译指令#include 来包含头文件。遇到编译指令#include 时，预处理器读取指定的文件，并将其插入到该编译指令所在的位置。在编译指令#include 中，不能使用通配符*或?来包含一组文件，但可以嵌套编译指令#include。换句话说，被包含的文件可以包含#include 编译指令，而该编译指令指定的文件也可以包含#include 编译指令，依此类推。大多数编译器都对嵌套深度有一定的限制，对于支持 ANSI 标准的编译器，通常嵌套深度可达 15 层。

在编译指令#include 中指定文件名的方式有两种。如果文件名用尖括号括起，如#include <stdio.h>，则预处理器将首先在标准目录中查找该文件。如果没有找到或没有指定标准目录，则编译器将在当前目录中查找。

什么是标准目录呢？在 DOS 中，是环境变量 INCLUDE 指定的目录。有关 DOS 环境的完整信息，请参阅 DOS 文档。通常，使用 SET 命令来设置环境变量（这通常是在 autoexec.bat 文件中设置的）。大多数编译器在安装时，将自动在 autoexec.bat 文件中设置 INCLUDE 变量。

另一种指定要包含的文件的方法是，用双引号将文件名括起：#include "myfile.h"。在这种情况下，预处理器将在被编译的源代码文件所在的目录（而不是标准目录）中查找。一般而言，您编写的头文件应保存在源代码文件所在的目录中，并使用双引号将其括起。而标准目录只用于保存编译器自带的头文件。

21.2.3 使用#if、#elif、#else 和#endif

这 4 个预处理器编译指令用于控制有条件的编译。有条件的编译意味着仅当特定的条件满足时，才对源代码块进行编译。在很多方面，预处理器编译指令#if 与 if 语句类似，区别在于，if 语句控制特定的语句是否执行，而#if 控制是否编译。

#if 语句块的结构如下：

```
#if condition_1
    statement_block_1
#elif condition_2
    statement_block_2
    ...
#elif condition_n
    statement_block_n
#else
    default_statement_block
#endif
```

#if 使用的测试表达式可以是结果为常量的任何表达式。不能使用 sizeof()运算符、强制类型转换或 float 类型。#if 最常用于检测#define 编译指令创建的符号常量。

其中每个 statement_block 由一条或多条语句组成，这些语句可以是任何类型的语句，包括预处理器编译指令。无需使用花括号将它们括起，虽然也可以这样做。

编译指令#if 和#endif 是必不可少的，而#elif 和#else 是可选的。可以有任意多个#elif 编译指令，但#else 只能有一个。编译器遇到编译指令#if 后，将检测相应的条件。如果为 TRUE（非零），则对#if 后面的语句进行编译；如果为 FALSE（零），则编译器依次检测每个#elif 编译指令中的条件，并对第一个条件为 TRUE 的#elif 编译指令中的语句进行编译。如果任何一个#elif 的条件都不为 TRUE，则对#else 编译指令后面的语句进行编译。

注意，#if...#endif 结构中的语句块最多有一个被编译。如果其中没有#else 编译指令，则可能没有任何语句被编译。

这些条件编译指令的用途只受您的想象力的限制。下面是一个例子。假设您要编写的程序将使用大量国家特有的信息，而每个国家的这种信息都被存储在一个头文件中，则要针对不同的国家编译该程序，可以使用下面这样的#if...#endif 结构：

www.plcworld.cn


```
#if ENGLAND == 1
#include "england.h"
#elif FRANCE == 1
#include "france.h"
#elif ITALY == 1
#include "italy.h"
#else
#include "usa.h"
#endif
```

然后，通过使用#define 来定义合适的符号常量，便可以控制在编译期间将哪个头文件包含进来。

21.2.4　使用#if...#endif 来帮助调试

#if...#endif 的另一种常见的用途是，用于包含有条件的调试代码。您可以定义一个名为 DEBUG 的符号常量，并将其设置为 0 或 1。然后，在程序中插入调试代码，如下所示：

```
#if DEBUG == 1
    debugging code here
#endif
```

在程序开发阶段，将 DEBUG 定义为 1，可将调试代码包含进来，帮助查找 bug。程序能够正常运行后，可以将 DEBUG 重新定义为 0，并重新编译程序，这样调试代码将不会被包含进来。

编写有条件的编译指令时，defined()运算符很有用。该运算符检查某个名称是否已被定义。因此下面的表达式随 NAME 是否被定义，而为 TRUE 或 FALSE：

```
defined(NAME)
```

通过使用 defined()，可以根据名称是否被定义对编译进行控制，而不管该名称的值是多少。对于前面的调试代码，可以将#if...#endif 重写为：

```
#if defined( DEBUG )
    debugging code here
#endif
```

也可以使用 defined()来这样做，即仅当某个名称没有被定义时，才定义它。可以像下面这样使用 NOT 运算符（!）：

```
#if !defined( TRUE )    /* if TRUE is not defined. */
#define TRUE 1
#endif
```

defined()运算符不要求名称被定义为特定的值。例如，下述代码定义了 RED，但并没有将它定义为某个特定的值：

```
#define RED
```

即使是这样，表达式 defined(RED)的值仍为 TRUE。当然，源代码中的 RED 将被删除，不用任何东西替换它，因此应小心使用这种定义方式。

21.2.5　避免将头文件包含多次

随着程序增大或使用头文件越来越频繁，很可能无意间包含一个头文件多次。这可能导致编译器感到迷惑，而停止编译。使用前面介绍的编译指令，可以避免这种情况发生。请看程序清单 21.5 中的程序。

程序清单 21.5　　prog.h：使用预处理器编译指令处理头文件

```
1:  /* prog.h - A header file with a check to prevent multiple includes! */
2:
3.  #if defined( prog_h )
4:  /* the file has been included already */
5:  #else
```

www.plcworld.cn


```
6:  #define prog_h
7:
8:  /* Header file information goes here... */
9:
10:
11:
12: #endif
```

分析：来看看该头文件做什么。第 3 行检查 prog_h 是否已被定义。prog_h 类似于该头文件的名称。如果 prog_h 已被定义，由于第 4 行为注释，因此程序将立刻转到#endif 处，因此不执行任何操作。

prog_h 是如何被定义的呢？是在第 6 行定义的。当该头文件首次被包含时，预处理器检查 prog_h 是否已被定义。没有，因此转到#else 语句处执行。在#else 后，首先定义了 prog_h，因此再包含该文件时，不会执行任何操作。第 7～11 行可以包含任意数目的命令或声明。



提示：创建自己的头文件时，一定要包含程序清单 21.5 那样的预编译器指令。这样，可以防止该头文件被包含多次。

21.2.6 #undef 编译指令

#undef 编译指令的功能与#define 相反，它撤销对名称的定义。下面是一个例子：

```
#define DEBUG 1
    /* In this section of the program, occurrences of DEBUG     */
    /* are replaced with 1, and the expression defined( DEBUG ) */
    /* evaluates to TRUE. *.
#undef DEBUG
    /* In this section of the program, occurrences of DEBUG  */
    /* are not replaced, and the expression defined( DEBUG ) */
    /* evaluates to FALSE. */
```

可以使用#undef 和#define 来创建只在源代码的某些部分被定义的名称。结合使用这种功能和#if 编译指令，可以更好地控制条件编译。

21.3 预定义的宏

大多数编译器都有大量预定义的宏，其中最有用的是__DATE__、__TIME__、__LINE__和__FILE__。这些宏前后分别有一个下划线，这是为了防止程序员重新定义它们。从理论上说，程序员不太可能创建名前前后有下划线的宏。

这些宏的工作原理与本章前面介绍的宏相同。预编译器将这些宏替换为相应的代码。__DATE__和__TIME__分别被替换为当前的日期和时间，这是源代码被预编译时的日期和时间。当程序有多个版本时，这些信息很有用。让程序显示其编译日期和时间，您可以知道运行的是否是最新的版本。

另外两个宏的用处更大。__LINE__将被替换为当前的行号，而__FILE__将被替换为当前源代码文件的名称。这两个宏最适合用于调试程序或处理错误。请看下面的 printf()语句：

```
31:
32: printf( "Program %s: (%d) Error opening file ", __FILE__, __LINE__ );
33:
```

如果这些代码位于名称 program.c 的程序中，则打印结果为：

```
Program myprog.c: (32) Error opening file
```

www.plcworld.cn

现在看起来这好像不太重要，但随着程序增大，被包含在多个源代码文件中时，查找错误将非常困难。使用__LINE__和__FILE__可使调试工作更容易。

应　该	不应该
定要使用__LINE__和__FILE__宏来使错误消息更有帮助。 一定要用圆括号将传递给宏的值括起，这样可以避免错误。例如，应这样： `#define CUBE(x) (x)*(x)*(x)` 而不应这样做： `#define CUBE(x) x*x*x`	使用#if 语句时，别忘了#endif。

21.4　使用命令行参数

C 程序能够访问通过命令行传递给它的参数。这里的参数指的是运行程序时，在程序名后输入的信息。例如，在运行名为 progname 的程序时，您可能在提示符 C:\>下输入：

```
C:\>progname smith jones
```

程序执行时，能够检索 smith 和 jones 这两个命令行。可以将这些信息看作是被传递给程序的 main()的参数。命令行参数使得可以在启动时（而不是运行期间）将信息传递给程序，有时候，这很方便。可以传递任意数目的命令行参数。只有在 main()内才能检索命令行参数，为此，可以这样声明 main()：

```
main(int argc, char *argv[])
{
    /* Statements go here */
}
```

第一个参数 argc 是一个整数，它指出有多少个命令行参数。这个值至少为 1，因为程序名被视为第一个参数。参数 argv[]是一个字符串指针数组，该数组的有效下标为 0 到 argc-1。指针 argv[0]指向程序名（包括路径信息）；argv[1]指向程序名后的第一个参数；依此类推。并不一定非得将参数命名为 argc 和 argv——您可以使用任何有效的变量名来接收命令行参数。然而，传统上使用这两个名称，因此应该遵循这种做法。

命令行中的参数由空白分隔，如果参数包含空格，则应使用双引号将整个参数括起。例如，在下面的命令中：

```
C:>progname smith "and jones"
```

第一个参数是 smith（argv[1]指向它），第二个参数是 and jones（argv[2]指向它）。程序清单 21.6 演示了如何在程序中访问命令行参数。

程序清单 21.6　　args.c：将命令行参数传递给 main()

```
1:  /* Accessing command-line arguments. */
2:
3:  #include <stdio.h>
4:
5:  int main(int argc, char *argv[])
6:  {
7:      int count;
8:
9:      printf("Program name: %s\n", argv[0]);
10:
11:     if (argc > 1)
```

www.plcworld.cn


```
12:    {
13:        for (count = 1; count < argc; count++)
14:            printf("Argument %d: %s\n", count, argv[count]);
15:    }
16:    else
17:        puts("No command line arguments entered.");
18:    return 0;
19: }
```

该程序的输出如下：

```
list21_6
Program name: C:\LIST2106.EXE
No command line arguments entered.

list2106 first second "3 4"
Program name: C:\LIST21_6.EXE
Argument 1: first
Argument 2: second
Argument 3: 3 4
```

分析：该程序只是打印用户输入的命令行参数而已。第 5 行使用的是前面介绍的 argc 和 argv 参数。第 9 行打印必不可少的命令行参数——程序名，它是 argv[0]。第 11 行检查是否有多个参数。为什么是有多个，而不是有呢？因为至少有一个参数——程序名。如果还有其他的参数，则 for 循环将每一个参数都打印到屏幕上（第 13 和 14 行）；否则，打印一条消息（第 17 行）。

命令行参数分为两类：必不可少的（没有它们，程序将无法运行）和可选的（如命令程序按特定的方式运行的标记）。例如，假设有一个对文件中的数据进行排序的程序。如果该程序从命令行接受文件名，则文件名是必不可少的。如果用户没有在命令行中输入文件名，程序必须采取某种方式进行处理。程序还可以接受一个/r 参数，该参数指出按反序排列。该参数不是必需的，如果有，程序按一种方式运行；如果没有，则按另一种方式运行。



注意：图形 IDE 通常允许您在对话框中输入命令行参数。例如，在 Dev-C++编译器中，可以通过单击编译对话框中的 Parameters 按钮，来输入命令行参数。图 21.5 是该按钮被单击后的编译对话框。



图 21.5 Parameters 按钮

应 该	不 应 该
在 main()中，应使用变量名 argc 和 argv 接收命令行参数。大多数 C 程序员都熟悉这两个名称。	不要假设用户将输入正确数目的命令行参数。应进行检查，如果不正确，则显示一条消息，告诉用户应该输入哪些参数。

www.plcworld.cn

21.5　总　结

今天的课程介绍了 C 编译器中的一些高级工具。您学习了如何将程序的源代码划分为多个文件（模块）。这种做法被称作模块化编程，可以很容易地在多个程序中重用通用函数。您还学习了如何使用预处理器编译指令来创建函数宏，以实现有条件的编译或其他任务。最后，介绍了编译器提供的一些函数宏。

21.6　问与答

问：编译多个文件时，编译器如何知道将哪个文件名作为可执行文件的名称？

答：您可能认为，编译器将使用包含 main()函数的文件的名称；情况并非总是如此。从命令行进行编译时，第一个文件将用于决定可执行文件的名称。例如，如果您使用 Borland 的 Turbo C 进行如下编译，则可执行文件的名称将为 File1.exe：

```
tcc file1.c main.c prog.c
```

问：头文件的扩展名必须是.h 吗？

答：不必。可以给头文件指定任何扩展名，但标准的做法是使用扩展名.h。

问：包含头文件时，是否可以指定路径？

答：可以。如果要指出被包含的文件所在的位置，可以这样做。在这种情况下，应使用引号将文件名括起。

问：今天是否介绍了所有的预定义宏和预处理器编译指令？

答：没有。今天介绍的只是大多数编译器中都有的预定义宏和预处理器编译指令。然而，大多数编译器还有其他的宏和常量。

问：使用 main()来接受命令行参数时，下述函数头也是可行的吗？

```
main( int argc, char **argv);
```

答：这个问题也许您自己就能回答。该声明使用一个指向字符指针的指针，而不是一个字符指针数组。由于数组也是指针，因此上述定义与本章介绍的几乎相同。这种声明也常用（更详细的信息，请参阅第 8 天的课程）。

21.7　作　业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

21.7.1　小测验

1. 模块化编程是什么意思？
2. 在模块化编程中，哪个是主模块？
3. 定义宏时，为何要用圆括号将每个参数括起？
4. 与函数相比，宏有何优缺点？
5. defined()运算符有何用途？
6. 使用#if，必须同时包含什么？
7. 编译后的 C 文件的扩展名是什么（假设已经链接）？
8. #include 有何功能？

www.plcworld.cn

9. 下述两行代码之间有何区别？

```
#include <myfile.h>
#include "myfile.h"
```

10. __DATE__有何用途？

11. argv[0]指向的是什么？

21.7.2 练习

下面的练习有多种解决方案，因此附录 F 没有提供它们的答案。

1. 使用您的编译器，将多个源代码文件编译成一个可执行文件（可以使用程序清单 21.1、21.2 和 21.3，也可以编写自己的程序清单）。

2. 编写一个处理错误的函数，它接受错误号、行号和模块名为参数。该函数打印一条格式化的错误消息，然后退出程序。使用预定义的宏来获得行号和模块名（从发现错误的地方将行号和模块名传递给函数）。下面是一个格式化错误消息的例子：

```
module.c (Line ##): Error number ##
```

3. 修改练习 2 中编写的函数，使错误消息更具描述性。使用编辑器创建一个文本文件，其中包含错误号和消息，并将其命名为 ERRORS.TXT。它可能包含这样的信息：

```
1    Error number 1
2    Error number 2
90   Error opening file
100  Error reading file
```

使处理错误的程序根据传递给它的错误号，在该文件中找到相应的消息，并显示它。

4. 编写模块化程序时，有些文件可能被包含多次。使用预处理器编译指令来编写这样一个头文件的框架：在编译期间，该头文件只在第一次遇到时被编译。

5. 编写一个程序，它接受两个文件名作为参数，并将第一个文件的内容复制到第二个文件中（有关如何使用文件的信息，请参阅第 16 天的课程）。

6. 选择一项您感兴趣、且能满足您的实际需求的编程任务。例如，您可以编写一个程序，对您收集的光盘进行分类、跟踪支票簿或计算有关购买房子的财务数据。要提高编程技能，并记住本书介绍的知识，唯一的途径就是去解决实际的编程问题。

www.plcworld.cn

第三周复习

至此，您完成了学习 C 语言编程的第三周也是最后一周的课程（别忘了附加课程）。本周首先介绍了诸如指针和磁盘文件等高级主题；接着介绍了大多数 C 编译器的函数库中都有的一些函数；最后介绍了充分利用编译器和 C 语言所需要了解的细节。下面的程序使用了很多有关这些主题的知识。



注意：左边的编号指出了代码涉及的主题是在哪一天介绍的，如果您对某些代码不太明白，可参阅相应的课程，以获取更详细的信息。

程序清单 R3.1　　week3.c：第三周复习的程序清单

```
         1: /* Program Name:  week3.c                                    */
         2: /* Program to keep track of names and phone numbers.        */
         3:
         4: /* Information is written to a disk file specified          */
         5: /* with a command-line parameter.                           */
         6:
         7: #include <stdlib.h>
         8: #include <stdio.h>
CH 19    9: #include <time.h>
CH 18   10: #include <string.h>
        11:
        12: /*** defined constants ***/
        13: #define YES   1
        14: #define NO    0
        15: #define REC_LENGTH  54
        16:
        17: /*** variables ***/
        18:
        19: struct record {
        20:    char fname[15+1];           /* first name + NULL      */
        21:    char lname[20+1];           /* last name + NULL       */
        22:    char mname[10+1];           /* middle name + NULL     */
        23:    char phone[9+1];            /* phone number + NULL    */
        24: } rec;
        25:
        26: /*** function prototypes ***/
        27:
CH 21   28: int  main(int argc, ch ar *argv[]);
        29: void display_usage(char *filename);
        30: int  display_menu(void);
```

www.plcworld.cn


```
          31: void get_data(FILE *fp, char *progname, char *filename);
          32: void display_report(FILE *fp);
          33: int  continue_function(void);
          34: int  look_up( FILE *fp );
          35:
          36: /* start of program */
          37:
CH 21     38: int main(int argc, ch ar *argv[])
          39: {
CH 16     40:    FILE *fp;
          41:    int  cont = YES;
          42:
CH 21     43:    if( argc < 2 )
          44:     {
          45:       display_usage("WEEK3");
          46:       exit(1);
          47:     }
          48:
          49:    /* Open file. */
CH 16     50:    if ((fp = fopen( argv[1], "a+")) == NULL)
          51:     {
          52:       fprintf( stderr, "%s(%d)—Error opening file %s",
          53:                        argv[0],__LINE__, argv[1]);
          54:       exit(1);
          55:     }
          56:
          57:    while( cont == YES )
          58:     {
          59:      switch( display_menu() )
          60:       {
CH 18     61:        case '1': get_data(fp, argv[0], argv[1]); /* Day 18*/
          62:                  break;
          63:        case '2': display_report(fp);
          64:                  break;
          65:        case '3': look_up(fp);
          66:                  break;
          67:        case '4': printf("\n\nThank you for using this Program!\n");
          68:                  cont = NO;
          69:                  break;
          70:        default:  printf("\n\nInvalid choice, Please select 1 to 4!");
          71:                  break;
          72:       }
          73:     }
CH 16     74:    fclose(fp);      /* close file */
          75:    return(0);
          76: }
          77:
          78: /* display_menu() */
          79:
          80: int display_menu(void)
          81: {
          82:    char ch, buf[20];
```

www.plcworld.cn


```
83:
84:     printf( "\n");
85:     printf( "\n     MENU");
86:     printf( "\n   ========\n");
87:     printf( "\n1.  Enter names");
88:     printf( "\n2.  Print report");
89:     printf( "\n3.  Look up number");
90:     printf( "\n4.  Quit");
91:     printf( "\n\nEnter Selection ==> ");
92:      gets(buf);
93:      ch = *buf;
94:      return(ch);
95: }
96:
97: /*********************************************************
98:   Function:  get_data()
99: *********************************************************/
100:
101: void get_data(FILE *fp, char *progname, char *filename)
102: {
103:    int cont = YES;
104:
105:    while( cont == YES )
106:     {
107:       printf("\n\nPlease enter information: " );
108:
109:       printf("\n\nEnter first name: ");
110:        gets(rec.fname);
111:       printf("\nEnter middle name: ");
112:        gets(rec.mname);
113:       printf("\nEnter last name: ");
114:        gets(rec.lname);
115:       printf("\nEnter phone in 123-4567 format: ");
116:        gets(rec.phone);
117:
CH 16  118:       if (fseek( fp, 0, SEEK_END ) == 0)
119:         if( fwrite(&rec, 1, sizeof(rec), fp) != sizeof(rec))
120:          {
121:            fprintf( stderr, "%s(%d) Error writing to file %s",
122:                        progname,__LINE__, filename);
123:            exit(2);
124:          }
125:       cont = continue_function();
126:     }
127: }
128:
129: /*********************************************************
130: Function:  display_report()
131: Purpose:   To print out the formatted names and numbers
132:            of people in the file.
133: *********************************************************/
134:
```

www.plcworld.cn


```
135: void display_report(FILE *fp)
136: {
137:     time_t rtime;
138:    int num_of_recs = 0;
139:
140:     time(&rtime);
141:
142:    fprintf(stdout, "\n\nRun Time: %s", ctime( &rtime));
143:    fprintf(stdout, "\nPhone number report\n");
144:
145:    if(fseek( fp, 0, SEEK_SET ) == 0)
146:     {
147:       fread(&rec, 1, sizeof(rec), fp);
148:       while(!feof(fp))
149:        {
150:         fprintf(stdout,"\n\t%s, %s %c %s", rec.lname,
151:                                rec.fname, rec.mname[0],
152:                                rec.phone);
153:         num_of_recs++;
154:         fread(&rec, 1, sizeof(rec), fp);
155:        }
156:       fprintf(stdout, "\n\nTotal number of records: %d",
157:             num_of_recs);
158:       fprintf(stdout, "\n\n* * * End of Report * * *");
159:     }
160:     else
161:       fprintf( stderr, "\n\n*** ERROR WITH REPORT ***\n");
162: }
163:
164: /*****************************************************
165: *  Function:  continue_function()
166: *****************************************************/
167:
168: int continue_function( void )
169: {
170:    char ch, buf[20];
171:    do
172:     {
173:       printf("\n\nDo you wish to enter another? (Y)es/(N)o ");
174:        gets(buf);
175:        ch = *buf;
176:    } while( strchr( "NnYy", ch) == NULL );
177:
178:    if(ch == 'n' || ch == 'N')
179:       return(NO);
180:     else
181:       return(YES);
182: }
183:
184: /*************************************************************
185: *  Function:  display_usage()
186: *************************************************/
```

www.plcworld.cn


```
187:
188: void display_usage( char *filename )
189: {
190:    printf("\n\nUSAGE: %s filename", filename );
191:    printf("\n\n   where filename is a file to store people\'s names");
192:    printf("\n     and phone numbers.\n\n");
193: }
194:
195: /************************************************
196: *  Function:  look_up()
197: *  Returns:   Number of names matched
198: ************************************************/
199:
200: int look_up( FILE *fp )
201: {
202:    char tmp_lname[20+1];
203:    int  ctr = 0;
204:
205:    fprintf(stdout, "\n\nPlease enter last name to be found: ");
206:    gets(tmp_lname);
207:
CH 17  208:    if( strlen(tmp_lname) != 0 )
209:     {
CH 16  210:      if (fseek( fp, 0, SEEK_SET ) == 0)
211:       {
CH 16  212:        fread(&rec, 1, sizeof(rec), fp);
213:        while( !feof(fp))
214:         {
CH 17  215:          if( strcmp(rec.lname, tmp_lname) == 0 )
216:          /* if matched */
217:           {
218:            fprintf(stdout, "\n%s %s %s - %s", rec.fname,
219:                                               rec.mname,
220:                                               rec.lname,
221:                                               rec.phone);
222:            ctr++;
223:           }
224:          fread(&rec, 1, sizeof(rec), fp);
225:         }
226:       }
227:      fprintf( stdout, "\n\n%d names matched.", ctr );
228:    }
229:    else
230:    {
231:      fprintf( stdout, "\nNo name entered." );
232:    }
233:   return(ctr);
234: }
```

分析：该程序在很多方面与第一周复习和第二周复习中的程序类似。记录的数据项更少，但功能更多。该程序使用户能够记录朋友、业务伙伴等的姓名和电话。该程序只记录姓、名、中名和电话，但要让它记录

www.plcworld.cn

其他信息也很容易，您可以将这项工作作为一个练习。该程序与前两个程序之间的主要区别在于，对可输入的人数没有限制，因为它使用磁盘文件来存储数据。

运行该程序时，在命令行中输入用于存储数据的文件的名称。main()从第 38 行开始，其中包含参数 argc 和 argv，用于获得命令行参数。有关这方面的知识，在第 21 天的课程中介绍过。第 43 行检查 argc，以确定在命令行中输入了多少个参数。如果 argc 小于 2，则说明用户只输入了一个参数（运行程序的命令），这意味着用户没有指定数据文件名。在这种情况下，程序将调用 display_usage()，并将 argv[0]作为参数传递给它。argv[0]是命令行中的第一个参数，它是程序的名称。

函数 display_usage()位于第 188-193 行。编写接受命令行参数的程序时，最好包含一个类似于 display_usage()的函数，用它来告诉用户如何使用该程序。该函数为何不直接使用程序名（week3），而要使用命令行参数呢？原因很简单。从命令行获取程序名时，无需担心用户可能将程序重命名，因为用法说明总是正确的。

该程序使用的大部分新概念是在第 16 天课程中介绍的。第 40 行声明了一个文件指针 fp，用于存取数据文件。第 50 行尝试以"a+"模式打开这个文件（arg[1]是第二个命令行参数——数据文件的名称）。之所以使用“a+”模式，是因为您希望能够将输入追加到该文件中，同时可以读取已有的记录。如果打开文件的操作失败，则第 52 和 53 行显示一条错误消息，然后第 54 行退出程序。错误消息中包含了描述性信息，而__LINE__（第 21 天课程中介绍过）指出发生错误的位置的行号。

如果成功地打开了文件，则显示一个菜单。如果用户选择退出程序，则第 74 行使用 fclose()关闭该文件，然后将控制权归还给操作系统。其他菜单选项让用户能够输入记录、显示所有的记录或查找特定的记录。

对于 get_data()函数，做了一些重要的修改。第 101 行是函数头。现在，该函数接受三个指针。其中第一个指针最重要：它是要写入的文件的句柄。第 105～126 行是一个 for 循环，该循环将不断执行，直到用户要求退出。第 107～116 行提示用户输入数据，格式与第二周复习的程序相同。第 118 行调用 fseek()，将磁盘文件的指针指向文件尾，以便追加新的信息。该程序没有对移动指针失败的情况进行处理。完整的程序应该处理这种故障，但为避免程序过长，这里没有这样做。第 119 行调用 fwrite()将数据写入到磁盘文件中。

该程序的报表功能也做了修改。大多数真正的报表的开头包含当前的日期和时间。第 137 行声明了一个名为 rtime 的变量。该变量被传递给 time()，然后使用 ctime()函数显示它。这些时间函数是在第 19 天的课程中介绍的。

程序打印文件中的记录之前，必须将文件指针重置到文件开头。这是在第 145 行通过调用 fseek()来完成的。重置文件指针后，便可以逐个地读取记录。第 147 行进行第一次读取。如果读取成功，则开始执行一个 while 循环，该循环将不断执行，直到到达文件尾（即 feof()返回一个非零值）。如果未到达文件尾，则第 150 行打印读取的信息，第 153 行对记录进行计数，而第 154 行尝试读取下一条记录。这里没有检查函数的返回值，旨在避免程序过长。要防止程序发生错误，调用函数时应进行检查，确保没有发生错误。

该程序中有一个函数是新添加的。第 200～234 行的 look_up()函数找出磁盘文件中包含特定姓的所有记录。第 205 和 206 行提示用户输入要查找的姓，并将其存储在一个名为 tmp_lname 的局部变量中。如果 tmp_lname 不为空（第 208 行），则将文件指针指向文件的开头。然后读取一条记录。第 215 行使用 strcmp()将记录的姓与 tmp_lname 进行比较。如果匹配，则打印该记录（第 218～222 行）。这一过程将不断重复下去，直到到达文件尾。同样，这里也没有检查函数的返回值，但您一定要检查。

您应该能够对这个程序进行修改，创建能够存储任何信息的文件。使用第三周介绍的函数以及 C 语言库中的其他函数，可以创建出能够完成任何工作的程序。

www.plcworld.cn

附加课程

经过21天的学习后，您应该熟悉C语言，并能够得心应手地创建自己的程序。

内容简介

大多数C语言学习者也想了解其他编程语言。附加课程将介绍另外三种编程语言：C++、Java和C#。虽然接下来的课程无法做全面的介绍，但将让您对这些语言有初步的了解。通过这几天的课程，您完全能够使用这些语言编写出功能完备的小型工程。

通过接下来的几天的课程，您会发现C++、Java和C#与C语言极其相似；同时还将了解它们不同于C语言的地方。附加课程1概要地介绍面向对象编程以及C++、Java和C#。最为重要的是，您将学习用于进行面向对象编程的关键结构（construct）。

附加课程2和3将介绍C++。C++是最流行的面向对象语言，它与C语言极其类似。事实上，C++是C语言的超集。编写C程序时使用的编译器，也可能可以用来编译C++程序。附加课程4～6介绍Java。Java与C和C++很相似，由于其移植性以及与Web的紧密关系，在过去的几年中Java非常流行。通过这几个附加课程，您将学习创建简单的小程序（applet）和应用程序。从附加课程4开始，您将学习有关Java的细节。

最后一个附加课程将对一种最新的面向对象编程语言—C#做简要的介绍。

具体内容见光盘。

www.plcworld.cn

附录 A　ASCII 字符集

十进制值	十六进制值	ASCII 字符	十进制值	十六进制值	ASCII 字符
0	00	空字符	29	20	空格
1	01	☺	30	21	!
2	02	●	31	22	"
3	03	♥	32	23	#
4	04	◆	33	24	$
5	05	♣	34	25	%
6	06	♠	35	26	&
7	07	•	36	27	'
8	08	◘	37	28	(
9	09	○	38	29	)
10	0A	■	39	2A	*
11	0B	♂	40	2B	+
12	0C	♀	41	2C	,
13	0D	♪	42	2D	-
14	0E	♫	43	2E	.
15	0F	☼	44	2F	/
16	10	►	45	30	0
17	11	◄	46	31	1
18	12	↕	47	32	2
19	13	‼	48	33	3
20	14	¶	49	34	4
21	15	§	50	35	5
22	16	▬	51	36	6
23	17	↨	52	37	7
24	18	↑	53	38	8
25	19	↓	54	39	9
26	1A	→	55	3A	:
27	1B	←	56	3B	;
28	1C	∟	57	3C	<

www.plcworld.cn

58	1D	↔	98	3D	=
59	1E	▲	99	3E	>
60	1F	▼	100	3F	?
61	40	@	101	60	`
62	41	A	102	61	a
63	42	B	103	62	b
64	43	C	104	63	c
65	44	D	105	64	d
66	45	E	106	65	e
67	46	F	107	66	f
68	47	G	108	67	g
69	48	H	109	68	h
70	49	I	110	69	i
71	4A	J	111	6A	j
72	4B	K	112	6B	k
73	4C	L	113	6C	l
74	4D	M	114	6D	m
75	4E	N	115	6E	n
76	4F	O	116	6F	o
77	50	P	117	70	p
78	51	Q	118	71	q
79	52	R	119	72	r
80	53	S	120	73	s
81	54	T	121	74	t
82	55	U	122	75	u
83	56	V	123	76	v
84	57	W	124	77	w
85	58	X	125	78	x
86	59	Y	126	79	y
87	5A	Z	127	7A	z
88	5B	[	128	7B	{
89	5C	\	129	7C	\|
90	5D	]	130	7D	}
91	5E	^	131	7E	~
92	5F	_	132	7F	Δ
93	80	Ç	133	A1	¡
94	81	ü	134	A2	ó
95	82	é	135	A3	ú
96	83	â	136	A4	ñ
97	84	ä	137	A5	Ñ

www.plcworld.cn

138	85	à	178	B2	▓
139	86	å	179	B3	│
140	87	ç	180	B4	┤
141	88	ê	181	B5	╡
142	89	ë	182	B6	╢
143	8A	è	183	B7	╖
144	8B	ï	184	B8	╕
145	8C	î	185	B9	╣
146	8D	ì	186	BA	║
147	8E	Ä	187	BB	╗
148	8F	Å	188	BC	╝
149	90	É	189	BD	╜
150	91	æ	190	BE	╛
151	92	Æ	191	BF	┐
152	93	ô	192	C0	└
153	94	ö	193	C1	┴
154	95	ò	194	C2	┬
155	96	û	195	C3	├
156	97	ù	196	C4	─
157	98	ÿ	197	C5	+
158	99	Ö	198	C6	╞
159	9A	Ü	199	C7	╟
160	9B	¢	200	C8	╚
161	9C	£	201	C9	╔
162	9D	¥	202	CA	╩
163	9E	₧	203	CB	╦
164	9F	ƒ	204	CC	╠
165	A0	á	205	CD	═
166	A6	ª	206	CE	╬
167	A7	º	207	CF	╧
168	A8	¿	208	D0	╨
169	A9	⌐	209	D1	╤
170	AA	¬	210	D2	╥
171	AB	½	211	D3	╙
172	AC	¼	212	D4	╘
173	AD	¡	213	D5	╒
174	AE	«	214	D6	╓
175	AF	»	215	D7	╫
176	B0	▒	216	D8	╪
177	B1	▒	217	D9	┘

www.plcworld.cn

218	DA	┌	237	ED	ø
219	DB	▉	238	EE	∈
220	DC	▄	239	EF	∩
221	DD	▌	240	F0	≡
222	DE	▐	241	F1	±
223	DF	▀	242	F2	≥
224	E0	α	243	F3	≤
225	E1	β	244	F4	⌠
226	E2	Γ	245	F5	⌡
227	E3	π	246	F6	÷
228	E4	Σ	247	F7	≈
229	E5	σ	248	F8	°
230	E6	μ	249	F9	•
231	E7	γ	250	FA	·
232	E8	Φ	251	FB	√
233	E9	θ	252	FC	n
234	EA	Ω	253	FD	2
235	EB	δ	254	FE	■
236	EC	∞	255	FF	

www.plcworld.cn

附录 B　C/C++中的保留字

表 B.1 列出了 C 语言的关键字和保留字，您不能在 C 程序中将它们用作其他用途。当然，它们可以位于双引号中。

这里还列出了 C++中（不是 C 语言中）的保留字，但没有对它们进行描述。如果您的 C 程序可能要移植到 C++中，则应避免使用这些保留字。

表 B.1　　C 语言的保留字

关 键 字	描　述
asm	指示内嵌的汇编语言代码
auto	默认存储类型，意思是变量在进入代码块时被创建，离开代码块时被释放
break	无条件地退出 for、while、switch 和 do…while 语句
case	用于 switch 语句中的命令
char	C 语言中最简单的数据类型
const	防止变量被修改的修饰符，参见 volatile
continue	立刻进入 for、while、do…while 循环的下一次迭代
default	用于 switch 语句中，处理不与任何 case 匹配的情况
do	与 while 一起使用的循环命令，循环将至少执行一次
double	用于存储双精度浮点值的数据类型
else	当 if 条件为 FALSE 时，用于执行另一条语句
enum	一种数据类型，这种类型的变量只能取某几个值
extern	指出变量将在其他地方被声明的限定符
float	用于存储浮点数的数据类型
for	循环命令，由初始化、递增和条件等三部分组成
goto	跳转到指定的位置
if	用于根据 TURE/FALSE 值来改变程序流程的命令
inline	用于将函数声明为内联的，内联函数调用将可能被替换为相应的函数代码，而不会像常规函数那样调用
int	用于存储整数的数据类型
long	用于存储大型整数的数据类型
register	存储限定符，指定应尽可能将变量存储在寄存器中
restrict	用于指针的访问限定符
return	导致离开当前函数，并返回到调用函数的命令，可用来返回一个值。
short	用于存储整数的数据类型。它不太常用，在大多数计算机上，其长度与 int 相同
signed	限定符，指出变量的值可以为正、也可以为负。参见 unsigned
sizeof	返回操作数长度（单位为字节）的运算符
static	告诉编译器应该保留变量的值

www.plcworld.cn

续表

struct	用于将数据类型不同的变量组合在一起
switch	用于改变程序流程，与 case 语句一起使用
typedef	用于给变量类型和函数类型提供别名
union	让多个变量共享内存空间
unsigned	指定变量只能包含正值，参见 unsigned
void	用于指定函数不返回任何值，还可用于指定通用类型的指针（即可以指向任何东西）
volatile	指定可以修改变量的值，参见 const
while	循环语句，只要条件为 TRUE 就执行一个代码段
_Bool	一种数据类型，其长度只够存储 0 或 1
_Complex	对复数类型的支持，编译器并不一定要支持_Complex
_Imaginary	对虚数类型的支持，编译器并不一定要支持_Imaginary

除了上述保留字外，C++还包含以下保留字：

catch	new	template
class	operator	this
delete	private	throw
except	protected	try
finally	public	virtual
friend		

www.plcworld.cn

附录 C　使用二进制和十六进制数

作为一名计算机程序员，有时候需要与二进制和十六进制的数字打交道。该附录将解释这些计数系统及其工作原理。为帮助理解，首先来复习一下常见的十进制。

C.1　十进制

十进制的基数为 10。这种计数系统中的数字，如 342 被表示为 10 的幂。第 1 位（最右边的一位）为 10 的 0 次方，第二位是 10 的一次方，依此类推。任何数的 0 次方都为 1，而任何数的一次方都是它自己。因此，对于 342：

3　$3\times10^2=3\times100=300$

4　$3\times10^1=3\times10=40$

2　$3\times10^0=3\times1=2$

总数为 342

十进制使用 10 个不同的数字：0～9。下述规则适用于十进制和其他任何计数系统：

- 数字被表示为基数的幂；
- 以 n 为基数的计数系统需要 n 个数。

下面介绍其他计数系统。

C.2　二进制

二进制的基数为 2，因此需要两个数：0 和 1。二进制对于计算机程序员很有帮助，因为它可用于表示数字式的开和关，而计算机芯片和内存正是以这样的方式工作的。下面是一个二进制数的例子，以垂直方式书写时，它为 1011,同时给出您更为熟悉的十进制表示。

1　$1\times2^3=1\times8=8$

0　$0\times2^2=0\times4=0$

1　$1\times2^1=1\times2=2$

1　$1\times2^0=1\times1=1$

总数为 11（十进制）

二进制有个缺点：表示大数时很繁琐。

C.3　十六进制

十六进制的基数为 16，因此需要 16 个数：0～9 和 A～F（表示 10～15）。下面是一个十六进制数的例子

www.plcworld.cn

（2DA），其对应的十进制数为：

2　$2\times16^2=3\times256=512$

D　$13\times16^1=13\times16=208$

A　$10\times16^0=10\times1=10$

总数为 730（十进制）

十六进制对于计算机工作而言很有用，因为它基于 2 的幂。十六进制数的每一位相当于二进制的四位，因此每两位相当于二进制的八位。表 D.1 列出了一些十六进制数及其对应的二进制和十进制数。

表 C.1　　十六进制数及对应的二进制和十进制数

十六进制数	十进制数	二进制数
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111
10	16	00010000
F0	240	11110000
FF	255	11111111

www.plcworld.cn

附录 D　移植性问题

移植性指的是将程序的源代码从一个平台移到另一个平台的难易程度。您为 PC 编写的程序可以在 UNIX 工作站上通过编译吗？在 Macintosh 机上呢？人们选择 C 语言的主要原因之一是其可移植性。C 语言可能是可移植性最高的编程语言之一。

为什么说可能呢？大多数 C 编译器都提供了扩展（extensions），扩展不是 C 标准的一部分，但很有用。如果您试图将使用了编译器扩展的程序移植到另一个平台，肯定会遇到问题，需要重新对程序的某些部分进行编码。如果您的程序绝不会被移植到另一个平台，则可以随便使用编译器扩展；但如果可能需要移植，则一定要谨慎地使用编译器扩展。本附录简要地讨论您需要考虑的问题。

D.1　ANSI 标准

移植性并非偶然发生的，它指的是您应遵循其他程序员和编译器遵循的一系列标准。因此，您应选择那些遵循 ANSI 制定的 C 语言编程标准的编译器。ANSI 委员会针对很多领域制定了标准，包括其他编程语言。几乎所有的 C 语言编译器都提供了遵循 ANSI 标准的选项。

D.2　ANSI 关键字

C 语言包含的关键字相对较少。关键字是保留用作程序命令的单词，第 1 天的课程和附录 B 都列出了 ANSI 保留字。

大多数编译器还提供了其他的保留字，如 near 和 huge。虽然多个编译器可能使用相同的编译器特定的关键字，但并不能保证这些关键字可以移植到所有的 ANSI 标准编译器中。

D.3　区分大小写

区分大小写是编程语言的一个重要问题。有些语言不区分大小写，但 C 语言是区分大小写的。这意味着变量 a 和 A 不是一回事。程序清单 D.1 说明了这种差别。

程序清单 D.1　　listD01.c：区分大小写

```
1:  /*=================================================*
2:   * Program: listD01.c                               *
3:   * Book:    Teach Yourself C in 21 Days             *
4:   * Purpose: This program demonstrates case sensitivity   *
5:   *=================================================*/
6:  #include <stdio.h>
```

www.plcworld.cn


```
7:  int main(void)
8:  {
9:   int     var1 = 1,
10:         var2 = 2;
11:    char  VAR1 = 'A',
12:         VAR2 = 'B';
13:    float Var1 = 3.3,
14:         Var2 = 4.4;
15:    int   xyz  = 100,
16:         XYZ  = 500;
17:
18:    printf( "\n\nPrint the values of the variables...\n" );
19:
20:    printf( "\nThe integer values:   var1 = %d, var2 = %d",
21:              var1, var2 );
22:    printf( "\nThe character values: VAR1 = %c, VAR2 = %c",
23:              VAR1, VAR2 );
24:    printf( "\nThe float values:     Var1 = %f, Var2 = %f",
25:              Var1, Var2 );
26:    printf( "\nThe other integers:   xyz = %d, XYZ = %d",
27:              xyz, XYZ );
28:
29:    printf( "\n\nDone printing the values!\n" );
30:
31:    return 0;
32: }
```

该程序的输出如下：

```
Print the values of the variables...

The integer values:    var1 = 1, var2 = 2
The character values:  VAR1 = A, VAR2 = B
The float values:      Var1 = 3.300000, Var2 = 4.400000
The other integers:    xyz = 100, XYZ = 500

Done printing the values!
```

分析：该程序使用了多个名称相同、但大小写不同的变量。第 9 和 10 行将 var1 和 var2 声明为 int 变量；第 11 和 12 行声明了变量 VAR1 和 VAR2；第 13 和 14 行将变量 Var1 和 Var2 声明为 float 变量。在声明这三组变量时，分别对其进行了初始化，以便后面可以打印它们——这是在第 20～25 行完成的。从输出可知，存储在变量中的值被保留下来，并被打印。

第 15 和 16 行声明了两个同名、同类型（int）的变量。这两个变量之间的唯一区别是：一个全部为大写，另一个为小写。并非所有安装了 C 语言编译器的计算机系统都是区分大小写的，因此使用只是大小写不同的变量的代码可能是不可移植的。要确保代码的可移植性，不应只通过大小写来区分变量。

区分大小写不仅仅会在编译器中导致问题，对于链接程序，这也可能导致问题。编译器可能能够区分只是大小写不同的变量，但链接程序可能不区分。

大多数编译器和链接程序都有一个用于设置是否区分大小写的标记。您应查看编译器文档，以确定要设置的标记。设置该标记后，当您对包含只是大小写不同的变量的代码进行编译时，将发生类似下面的错误。当然，其中 var1 被替换为您使用的只是大小写不同的变量的名称。

```
listD01.c:
Error listD01.c 16: Multiple declaration for 'var1' in function main
```

www.plcworld.cn


```
*** 1 errors in Compile ***
```

D.4 可移植的字符

在计算机中，字符是用数字表示的。在 IBM PC 及其兼容机中，字母 A 是用数字 65 表示的，则字母 a 被表示为 97。这些数字来自 ASCII 字符表（参见附录 A）。

编写可移植的程序时，不能假设计算机系统将使用 ASCII 表作为字符转换表。使用的字符表可能随计算机系统而异。换句话说，在大型机上，数字 65 表示的可能不是字符 A。



警告：使用字符数值时应谨慎，因为它可能是不可移植的。

在字符集如何定义方面，有两条通用的规则。第一条规则是，字符值的取值范围不能大于 char 类型的取值范围。在 8 位系统中，char 变量能够存储的最大值为 255。因此，字符的值不能大于 255；而在 16 位的系统中，字符的值不能超过 65535。

第二条规则是，每个字符都必须用一个正数表示。在 ASCII 字符集中，值为 1～127 的字符是可移植的。而值为 128～255 的字符不一定是可移植的，这是因为对于有符号 char 变量，其可能的取值中只有 127 个为正。

D.5 确保 ANSI 兼容性

预定义的常量__STDC__用于帮助确保 ANSI 兼容性。当编译程序时，如果设置了 ANSI 兼容性，则该常量将被定义——通常被定义为 1；如果没有设置 ANSI 兼容性，则该常量将不会被定义。

几乎所有的编译器都提供了用于执行 ANSI 标准的选项。通常，在 IDE（集成开发环境）中，这可以通过设置开关来完成；而对于命令行编译器，可在编译时传递一个额外的参数来实现。通过设置 ANSI 兼容型选项，可以确保程序能被移植到其他编译器和平台中。

对于 Borland C++编译器，可以使用下面的命令来编译程序，使之与 ANSI 兼容：

```
BCC -A program.c
```

而对于 Microsoft 编译器，可以执行下面的命令：

```
CL /Ze program.c
```



注意：大多数集成开发环境（IDE）编译器都提供了一个 ANSI 选项，选中该选项可以确保程序的 ANSI 兼容性。

这样，编译器将进行额外的错误检查，确保遵循了 ANSI 规则。有时候，可能不再检查某些错误和警告，如原型检查。如果使用函数之前没有提供其原型，则大多数编译器会发出警告；但 ANSI 标准不要求这样。因为 ANSI 不要求原型，因此您不会收到必须有原型的警告。

D.6 绕开 ANSI 标准

不想在编译程序时设置 ANSI 兼容性的原因有多种。最常见的原因是，您想充分利用编译器添加的特性。很多特性（如特殊的屏幕处理函数）是 ANSI 标准中没有的或者是编译器特有的。如果要使用编译器特有的特性，则不应设置 ANSI 标记。另外，使用这些特性后，程序将是不可移植的。本附录的后面将介绍一种绕

www.plcworld.cn

开这种限制的方式。

应　该	不应该
应不仅仅通过大小写来区分不同的变量。	不要对字符的数字值做任何假设。

D.7　使用可移植的数值变量

对于特定类型的变量而言，可存储的数值可能随编译器而异。对于每种变量类型可存储的数值，ANSI 标准定义了一些规则。第 3 天的课程中，表 3.1 列出了在 IBM-兼容 PC 中，各种数据类型可存储的值，但在其他系统中，可存储的值不一定与此相同。

对于变量类型可存储的值，以下规则适用：

- char 是最小的数据类型，一个 char 变量占用一个字节的内存；
- short 变量占用的内存不比 int 变量多；
- int 变量占用的内存不比 long 变量多；
- unsigned int 变量占用的内存不比 int 多；
- float 变量占用的内存不比 double 变量多。

程序清单 D.2 打印变量占用的内存量，打印的值取决于该程序是在什么样的机器上被编译的。

程序清单 D.2　　打印各种数据类型的变量占用的内存量

```
1:  /*=================================================*
2:   * Program: listD02.c                               *
3:   * Book:    Teach Yourself C in 21 Days             *
4:   * Purpose: This program prints the sizes of the variable  *
5:   *          types of the machine the program is compiled on *
6:   *=================================================*/
7:  #include <stdio.h>
8:  int main(void)
9:  {
10:    printf( "\nVariable Type Sizes" );
11:    printf( "\n======================" );
12:    printf( "\nchar           %d", sizeof(char) );
13:    printf( "\nshort          %d", sizeof(short) );
14:    printf( "\nint            %d", sizeof(int) );
15:    printf( "\nfloat          %d", sizeof(float) );
16:    printf( "\ndouble         %d", sizeof(double) );
17:
18:    printf( "\n\nunsigned char   %d", sizeof(unsigned char) );
19:    printf( "\nunsigned short  %d", sizeof(unsigned short) );
20:    printf( "\nunsigned int    %d\n", sizeof(unsigned int) );
21:
22:    return 0;
23: }
```

该程序的输出如下：

```
Variable Type Sizes
==========================
char           1
short          2
```

www.plcworld.cn


```
int             2
float           4
double          8

unsigned char   1
unsigned short  2
unsigned int    2
```

分析：该程序使用运算符 sizeof()来打印每种变量类型占用的内存量（单位为字节）。上述输出是使用 16 位编译器来编译该程序，并在 16 位的 IBM-兼容 PC 上运行它时得到的。输出可能随编译器和系统而异。例如，如果使用的是 32 位的编译器，并在 32 位的机器上运行该程序，则 int 变量占用的内存将为 4 字节，而不是 2 字节。

D.7.1 最大值和最小值

如果变量占用的内存量随机器而变，那么如何知道该变量能够存储哪些值呢？这取决于变量占用的内存量以及它是有符号还是无符号的。表 3.2 列出了变量可存储的值与其占用的字节数之间的关系。对于整型变量而言，可存储的最大和最小值取决于变量的位数；而对于浮点型变量（如 float 和 double），可以通过降低精度，存储更大的值。表 D.1 列出了整型变量和浮点型变量可存储的最大和最小值。

表 D.1　　可能的取值与变量占用的字节数之间的关系

字节数	最大无符号值	最小有符号值	最大有符号值
整型			
1	255	-128	127
2	65535	-32768	32767
4	4294967295	-2147483648	2147483647
8		1.84467XE19	
浮点型			
4*		3.4E-38	3.4E38
8**		1.7E-308	1.7E308
10***		3.4E-4932	3.4E4932

*精度为 7 位小数

**精度为 15 位小数

***精度为 19 位小数

虽然根据变量的类型和占用的字节数可以知道它能够存储的最大值，但正如前面指出的，在可移植的程序中，变量占用的字节数并非总是已知的。另外，您无法确定浮点数的精度。因此，在将值赋给变量时一定要小心。例如，将 3000 赋给一个 int 变量是安全的，但将 100000 赋给 int 变量呢？如果是在 16 位的机器上，且该变量为 unsigned int，则结果将是异常的，因为其最大取值为 65535。如果 int 变量占 4 个字节，则这种赋值操作不会导致任何问题。



警告：并非在所有的编译器中，表 D.1 列出的关系都是正确的。这种关系可能随编译器而稍有不同，尤其是对浮点型而言，因为它们的精度可能不同。表 D.2 和 D.3 提供了一种兼容地使用这些数字的方式。

ANSI 对一组定义的常量进行了标准化，这些常量包含在头文件 limits.h 和 float.h 中。这些常量定义了不同类型的变量的位数及其最大和最小值。表 D.2 列出了 limits.h 中定义的常量，这些常量适用于整型；而 float.h 中定义的常量适用于浮点型。

www.plcworld.cn

表 D.2　limits.h 中定义的 ANSI 常量

常　量	值
CHAR_BIT	字符变量的位数
CHAR_MIN	字符变量的最小值（有符号）
CHAR_MAX	字符变量的最大值（有符号）
SCHAR_MIN	有符号字符变量的最小值
SCHAR_MAX	有符号字符变量的最大值
UCHAR_MAX	无符号字符变量的最大值
INT_MIN	int 变量的最小值
INT_MAX	int 变量的最大值
UINT_MAX	无符号 int 变量的最大值
SHRT_MIN	short 变量的最小值
SHRT_MAX	short 变量的最大值
USHRT_MAX	无符号 short 变量的最大值
LONG_MIN	long 变量的最小值
LONG_MAX	long 变量的最大值
ULONG_MAX	无符号 long 变量的最大值
LLONG_MAX	long long 变量的最大值
ULLONG_MAX	无符号 long long 变量的最大值

表 D.3　float.h 中定义的 ANSI 常量

常　量	值
FLT_DIG	float 变量的精确位数
DBL_DIG	double 变量的精确位数
LDBL_DIG	long double 变量的精确位数
FLT_MAX	float 变量的最大值
FLT_MAX_10_EXP	float 变量的最大指数值（以 10 为底）
FLT_MAX_EXP	float 变量的最大指数值（以 2 为底）
FLT_MIN	float 变量的最小值
FLT_MIN_10_EXP	float 变量的最小指数值（以 10 为底）
FLT_MIN_EXP	float 变量的最小指数值（以 2 为底）
DBL_MAX	double 变量的最大值
DBL_MAX_10_EXP	double 变量的最大指数值（以 10 为底）
DBL_MAX_EXP	double 变量的最大指数值（以 2 为底）
DBL_MIN	double 变量的最小值
DBL_MIN_10_EXP	double 变量的最小指数值（以 10 为底）
DBL_MIN_EXP	double 变量的最小指数值（以 2 为底）
LDBL_MAX	long double 变量的最大值
LDBL_MAX_10_EXP	long double 变量的最大指数值（以 10 为底）
LDBL_MAX_EXP	long double 变量的最大指数值（以 2 为底）
LDBL_MIN	long double 变量的最小值
LDBL_MIN_10_EXP	long double 变量的最小指数值（以 10 为底）
LDBL_MIN_EXP	long double 变量的最小指数值（以 2 为底）

www.plcworld.cn

存储数字时，可以使用表 D.2 和 D.3 中的值。确保存储的值不小于最小常量，且不大于最大常量可以保证程序是可移植的。程序清单 D.3 打印了 ANSI 定义的常量的值；程序清单 D.4 演示了如何使用这些常量。程序的输出可能随使用的编译器而异。

程序清单 D.3　　打印 ANSI 定义的常量的值

```
1:  /*=============================================================== *
2:   * Program:  listD03.c                                          *
3:   * Book:     Teach Yourself C in 21 Days                        *
4:   * Purpose:  Display of defined constants.                      *
5:   *=============================================================== */
6:  #include <stdio.h>
7:  #include <float.h>
8:  #include <limits.h>
9:
10:  int main( void )
11:  {
12:      printf( "\n CHAR_BIT      %d ", CHAR_BIT );
13:      printf( "\n CHAR_MIN      %d ", CHAR_MIN );
14:      printf( "\n CHAR_MAX      %d ", CHAR_MAX );
15:      printf( "\n SCHAR_MIN     %d ", SCHAR_MIN );
16:      printf( "\n SCHAR_MAX     %d ", SCHAR_MAX );
17:      printf( "\n UCHAR_MAX     %d ", UCHAR_MAX );
18:      printf( "\n SHRT_MIN      %d ", SHRT_MIN );
19:      printf( "\n SHRT_MAX      %d ", SHRT_MAX );
20:      printf( "\n USHRT_MAX     %d ", USHRT_MAX );
21:      printf( "\n INT_MIN       %d ", INT_MIN );
22:      printf( "\n INT_MAX       %d ", INT_MAX );
23:      printf( "\n UINT_MAX      %ld ", UINT_MAX );
24:      printf( "\n LONG_MIN      %ld ", LONG_MIN );
25:      printf( "\n LONG_MAX      %ld ", LONG_MAX );
26:      printf( "\n ULONG_MAX     %e ", ULONG_MAX );
27:      printf( "\n FLT_DIG       %d ", FLT_DIG );
28:      printf( "\n DBL_DIG       %d ", DBL_DIG );
29:      printf( "\n LDBL_DIG      %d ", LDBL_DIG );
30:      printf( "\n FLT_MAX       %e ", FLT_MAX );
31:      printf( "\n FLT_MIN       %e ", FLT_MIN );
32:      printf( "\n DBL_MAX       %e ", DBL_MAX );
33:      printf( "\n DBL_MIN       %e \n", DBL_MIN );
34:
35:      return(0);
36:  }
```

该程序的输出如下：

```
 CHAR_BIT      8
 CHAR_MIN      -128
 CHAR_MAX      127
 SCHAR_MIN     -128
 SCHAR_MAX     127
 UCHAR_MAX     255
 SHRT_MIN      -32768
 SHRT_MAX      32767
```

www.plcworld.cn


```
USHRT_MAX       65535
INT_MIN         -2147483648
INT_MAX         2147483647
UINT_MAX        -1
LONG_MIN        -2147483648
LONG_MAX        2147483647
ULONG_MAX       8.121967e-298
FLT_DIG         6
DBL_DIG         15
LDBL_DIG        15
FLT_MAX         3.402823e+038
FLT_MIN         1.175494e-038
DBL_MAX         1.797693e+308
DBL_MIN         2.225074e-308
```



注意：该程序的输出随编译器而异，因此您运行该程序时，输出可能与此不同。

分析：该程序很简单，它由 printf()函数调用组成。每个函数调用都打印一个常量，其中使用的转换字符随打印的值的类型而异。该程序指出了编译器使用的值。您也可以查看头文件 float.h 和 limits.h，看其中定义了这些值没有。该程序使得确定常量的值很容易。

程序清单 D.4　　使用 ANSI 定义的常量

```
1:  /*=================================================================*
2:   * Program: listD04.c                                              *
3:   * Book:    Teach Yourself C in 21 Days                            *
4:   *                                                                 *
5:   * Purpose: To use maximum and minimum constants.                  *
6:   * Note:    Not all valid characters are displayable to the        *
7:   *          screen!                                                *
8:   *=================================================================*/
9:
10:  #include <float.h>
11:  #include <limits.h>
12:  #include <stdio.h>
13:
14:  int main( void )
15:  {
16:     unsigned char ch;
17:     int  i;
18:
19:     printf( "Enter a numeric value.");
20:     printf( "\nThis value will be translated to a character.");
21:     printf( "\n\n==> " );
22:
23:     scanf("%d", &i);
24:
25:     while( i < 0 || i > UCHAR_MAX )
26:     {
27:        printf("\n\nNot a valid value for a character.");
28:        printf("\nEnter a value from 0 to %d ==> ", UCHAR_MAX);
```

www.plcworld.cn


```
29:
30:         scanf("%d", &i);
31:      }
32:      ch = (char) i;
33:
34:      printf("\n\n%d is character %c\n", ch, ch );
35:
36:      return(0);
37:  }
```

该程序的运行情况如下：

```
Enter a numeric value.
This value will be translated to a character.
==> 5000
Not a valid value for a character.
Enter a value from 0 to 255 ==> 69

69 is character E
```

分析：该程序清单演示了如何使用常量 UCHAR_MAX。其中第一项新的内容是第 10 和 11 行。正如前面指出的，这两个头文件中包含了定义的常量。您可能会问，为何要包含头文件 float.h（第 10 行）？由于没有使用任何浮点常量，因此可以不包含头文件 float.h。然而，第 11 行是必不可少的。该头文件中包含了程序后面使用的 UCHAR_MAX 的定义。

第 16 和 17 行声明了程序要使用的变量：一个 unsigned char 变量（ch）和一个 int 变量（i）。声明变量后，使用几条打印语句来提示用户输入一个数字。用户输入的数字被存储到变量 i 中。由于 int 变量能够存储较大的数字，因此使用这种变量来存储用户的输入。如果使用 char 变量来存储，则如果用户输入的值过大，将对其进行转换，使之能够被存储到 char 变量中。为演示这一点，可将第 23 行的 i 改为 ch。

第 25 行使用定义的常量来判断输入的值是否大于 unsigned char 变量的最大值。这里判断输入的值是否大于 unsigned char 变量的最大值，而不是 int 变量的最大值，因为程序要打印的是字符，而不是整数。如果输入的值不对应于一个字符——准确地说，是无法存储在 unsigned char 变量中，则告诉用户可以输入哪些值（第 28 行），然后让用户再输入一个有效的值。

第 32 行将整数强制转换为字符值。在更复杂的程序中，您可能发现，使用 char 变量而不是 int 变量更容易。这样，无需将对 char 变量而言无效的值转换为 int。对于该程序而言，打印字符的代码行（第 34 行）可以使用 i，而不是 ch。

D.7.2 确定数字的类型

有些情况下，需要了解有关变量的信息。例如，您可能需要知道它是数值、控制字符、大写字母还是其他类型。确定类型的方式有两种。程序清单 D.5 演示了一种确定 char 变量的值是否为字母的方式。

程序清单 D.5　　确定 char 变量的值是否为字母

```
1:  /*=======================================================*
2:   * Program: listD05.c
3:   * Purpose: This program may not be portable due to the  *
4:   *          way it uses character values.                *
5:   *=======================================================*/
6:  #include <stdio.h>
7:  int main(void)
8:  {
```

www.plcworld.cn


```
9:    unsigned char x = 0;
10:   char trash[256];              /* used to remove extra keys */
11:   while( x != 'Q' && x != 'q'
12:   {
13:      printf( "\n\nEnter a character (Q to quit) ==> " );
14:
15:      x = getchar();
16:
17:      if( x >= 'A' && x <= 'z'
18:      {
19:         printf( "\n\n%c is a letter of the alphabet!", x );
20:         printf("\n%c is an uppercase letter!", x );
21:      }
22:      else
23:      {
24:         if( x >= 'a' && x <= 'z'
25:         {
26:            printf( "\n\n%c is a letter of the alphabet!", x );
27:            printf("\n%c is a lowercase letter!", x );
28:         }
29:         else
30:         {
31:            printf( "\n\n%c is not a letter of the alphabet!", x );
32:         }
33:      }
34:      gets(trash); /* eliminates enter key */
35:   }
36:   printf("\n\nThank you for playing!\n");
37:   return 0;
38: }
```

该程序的运行情况如下：

```
Enter a character (Q to quit) ==> A

A is a letter of the alphabet!
A is an uppercase letter!

Enter a character (Q to quit) ==>f

f is a letter of the alphabet!
f is a lowercase letter!

Enter a character (Q to quit) ==> 1

1 is not a letter of the alphabet!

Enter a character (Q to quit) ==> *

* is not a letter of the alphabet!

Enter a character (Q to quit) ==> q
```

www.plcworld.cn


```
q is a letter of the alphabet!
q is a lowercase letter!

Thank you for playing!
```

分析：该程序检查 char 变量 x 的值是否在 A 和 Z 之间或 a 和 z 之间。如果是，则可以认为它是一个字母。这是一个糟糕的假设！对于字符集中字符的排列次序，并没有任何标准。如果您使用的是 ASCII 字符集，这种判断方式不会有问题，但程序不一定是可移植的。要确保可移植性，应使用一个判断字符类型的函数。

有多个用于判断字符类型的函数，表 D.4 列出这些函数及其用途。如果给定的字符不符合检查标准，这些函数返回 0；否则返回一个非零值。

表 D.4　　判断字符类型的函数

函　数	描　述
isalnum()	判断字符是否为字母或数字
isalpha()	判断字符是否为字母
iscntrl()	判断字符是否为控制字符
isdigit()	判断字符是否为十进制数字
isgraph()	判断字符是否为可打印的（空格除外）
islower()	判断字符是否为小写字母
isprint()	判断字符是否为可打印的
ispunct()	判断字符是否为标点符号
isspace()	判断字符是否为空白字符
isupper()	判断字符是否为大写字母
isxdigit()	判断字符是否为十六进制数字

除了检查两个字符是否相等外，不应对字符的值进行比较。例如，您可以检查一个 char 变量的值是否为'A'，但不要检查它的值是否大于'A'。

```
if( X > 'A')  /*NOT PORTABLE!! */
...
if( X == 'A')  /*PORTABLE */
...
```

程序清单 D.6 对 D.5 进行了修改，它使用字符类型判断函数，而不是进行范围检查，因此可移植性高得多。

程序清单 D.6　　使用字符类型判断函数

```
1:  /*=================================================================*
2:   * Program: listD06.c                                              *
3:   * Book:    Teach Yourself C in 21 Days                            *
4:   * Purpose: This program is an alternative approach to             *
5:   *          the same task accomplished in Listing D.5.             *
6:   *          This program has a higher degree of portability!       *
7:   *=================================================================*/
8:  #include <ctype.h>
9:
10:  int main(void)
11:  {
12:    unsigned char x = 0;
13:    char trash[256];              /* use to flush extra keys */
14:    while( x != 'Q' && x != 'q'
15:    {
16:      printf( "\n\nEnter a character (Q to quit) ==> " );
```

www.plcworld.cn


```
17:
18:     x = getchar();
19:
20:     if( isalpha(x) )
21:     {
22:       printf( "\n\n%c is a letter of the alphabet!", x );
23:       if( isupper(x) )
24:       {
25:         printf("\n%c is an uppercase letter!", x );
26:       }
27:       else
28:       {
29:         printf("\n%c is a lowercase letter!", x );
30:       }
31:     }
32:     else
33:     {
34:       printf( "\n\n%c is not a letter of the alphabet!", x );
35:     }
36:     gets(trash);   /* get extra keys */
37:   }
38:   printf("\n\nThank you for playing!\n");
39:   return(0);
40: }
```

该程序的运行情况如下：

```
Enter a character (Q to quit) ==> z

z is a letter of the alphabet!
z is a lowercase letter!

Enter a character (Q to quit) ==> T

T is a letter of the alphabet!
T is an uppercase letter!

Enter a character (Q to quit) ==> #

# is not a letter of the alphabet!

Enter a character (Q to quit) ==> 7

7 is not a letter of the alphabet!

Enter a character (Q to quit) ==> Q

Q is a letter of the alphabet!
Q is an uppercase letter!

Thank you for playing!
```

分析：该程序的运行情况与程序清单D.5几乎相同——假设您输入的值相同。该程序使用字符类型判断

www.plcworld.cn

函数，而不是对范围进行检查。第 8 行将头文件 ctype.h 包含进来，以便可以使用字符类型判断函数。第 20 行使用函数 isalpha()判断用户输入的是否为字母。如果是，则第 22 行打印一条消息，指出这一点。第 23 行使用 isupper()函数判断字符是否是大小字母。如果是，则执行第 25 行，打印一条消息；否则，执行第 29 行，打印另一条消息。如果用户输入的字符不是字母，则执行 34 行，打印一条消息。该程序使用一个 while 循环（从第 14 行开始），在用户输入 Q 或 q 之前，该程序将不断执行。您可能认为，第 14 行将导致该程序不可移植。这种想法是错误的，因为对字符进行相等判断是可移植的，但其他比较则是不可移植的。“不等于”和“等于”都属于相等判断。

应　该	不应该
应尽可能使用字符类型判断函数。 请记住，!=属于相等判断。	编写可移植的程序时，要判断变量的最大可能取值，不应使用数值，而应使用定义的常量。

D.7.3　转换字符的大小写：一个可移植性范例

编程时，常常需要转换字符的大小写。为此，很多人编写一个下面这样的函数：

```
char conv_to_upper( char x )
{
    if( x >= 'a' && x <= 'z')
    {
        x -= 32;
    }
    return( x )
}
```

正如前面指出的，上述 if 语句可能是不可移植的。下面是修改后的函数，它在 if 语句中使用的是前一节介绍的可移植函数：

```
char conv_to_upper( char x )
{
    if( isalpha( x ) && islower( x ) )
    {
        x -= 32;
    }
    return( x )
}
```

就移植性而言，这个函数优于前一个；但它仍然不是完全可移植的。该函数假设小写字母的值比大写字母大 32。对于 ASCII 字符集而言，情况确实如此。在 ASCII 字符集中，‘A’ + 32 等于‘a’；但并非在所有系统中情况都如此，尤其是在非 ASCII 字符系统中。

有两个 ANSI 标准函数可用于转换字符的大小写。函数 toupper()将小写字母转换为大写；而 lowercase()函数将大写字母转换为小写。因此，不用自己编写上述转换函数，而应使用函数 toupper()。

正如您看到的，存在转换大小写的函数。另外，这些函数是由 ANSI 标准定义的，因此是可移植的。

D.8　可移植的结构和共用体

如果程序需要移植，则使用结构和共用体时也应小心。使用这些元素时，可能发生不兼容问题的地方有两个：字对齐和成员的存储顺序。

D.8.1　字对齐

字对齐（word alignment）对于结构的兼容性至关重要。字对齐指的是将数据与字的边界对齐。字是一组

www.plcworld.cn

字节。字长通常等于计算机的处理器使用的长度。例如，在 16 位的 IBM PC 上，字长通常为两个字节。两个字节为 16 位。

为帮助理解，请看一个例子。对于下面的结构，可以依据 int 成员占两个字节以及 char 成员占一个字节来确定它需要多少个字节的存储空间。

```
struct struct_tag {
  int    x;    /* ints will be 2 bytes */
  char   a;    /* chars are 1 byte */
  int    y;
  char   b;
  int    z;
} sample = { 100, 'A', 200, 'B', 300};
```

将 int 成员占用的空间和 char 成员占用的空间相加，可以知道总存储空间为 8 字节。这个答案是正确的，也可能是错误的！如果启用了字对齐，该结构将占用 10 个字节的存储空间。图 D.1 和 D.2 说明了该结构在内存中是如何被存储的。

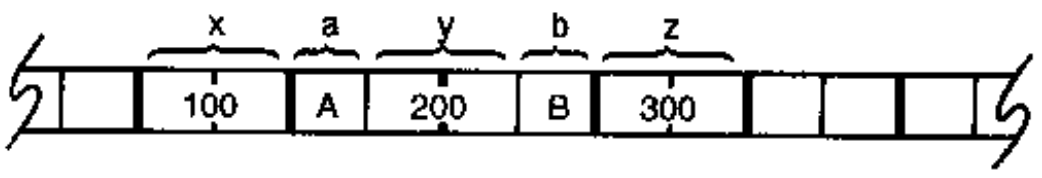


图 D.1　禁用了字对齐

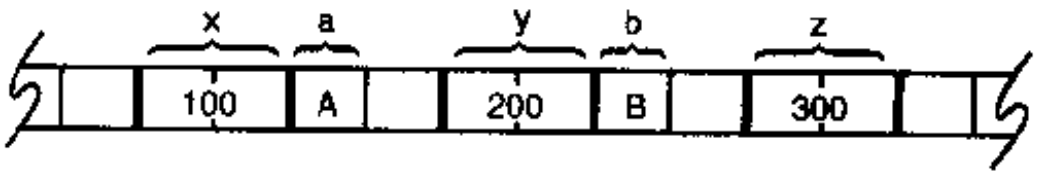


图 D.2　启用了字对齐

程序不能假设字对齐是启用了还是禁用了。您无法知道成员将与两个字节、4 个字节还是 8 个字节的字的边沿对齐。

D.8.2　读写结构

读写结构时必须小心。最好不要通过字面常量来指定结构或共用体的长度。将结构写入到文件或从文件中读取结构时，文件可能是不可移植的。这意味着您只需使程序是可移植的，程序必须能够读写它被编译时，所在机器的特有的数据文件。下面是一个可移植的 read 语句的例子：

```
fread( &the_struct, sizeof( the_struct ), 1, filepointer );
```

正如您看到的，这里使用的是 sizeof 运算符，而不是字面值。不管字对齐是启用还是禁用的，这都将读取正确数目的字节。

1. 成员的次序

创建结构时，您可能假设其成员将按列出次序被存储。并没有某种标准规定成员的存储次序，因此您不能对成员的存储顺序做任何假设。

2. 预处理器编译指令

第 21 天的课程介绍了多个可以使用的预处理器编译指令。ANSI 标准定义了多个预处理器编译指令，其中经常使用的有两个：#include 和#define。表 D.5 列出了 ANSI 标准定义的预处理器编译指令。

www.plcworld.cn

表 D.5　　ANSI 标准定义的预处理器编译指令

#define	#if
#elif	#ifdef
#else	#ifndef
#endif	#include
#error	#pragma

3. 使用预定义的常量

每个编译器都自带一些预定义的常量，其中的大部分都是编译器特定的。这意味着它们可能是不可移植的；然而，ANSI 标准也定义了几个预定义的常量，包括：

- __DATE__：将被替换为程序编译时的日期，这是一个字面字符串（用双引号括起的文本）。其格式为“Mmm DD, YYYY”，例如，1998 年 1 月 1 日被表示为“Jan 1, 1998”。
- __FILE__：将被替换为源代码文件的名称，是一个字面字符串。
- __LINE__：将被替换为__LINE__所在行的行号，是一个十进制数字。
- __STDC__：如果按 ANSI 标准编译文件，则该常量被定义为 1；否则，未被定义。
- __TIME__：将被替换为程序的编译时间，是一个字面字符串（用双引号括起的文本）。其格式为“HH:MM:SS”，例如“12:15:03”。

D.8.3　在可移植的程序中使用非-ANSI 特性

程序即使使用了不是 ANSI 定义的常量和其他命令，也可以是可移植的。为此，只需确保仅当编译器支持使用的特性时，才使用常量。大多数编译器提供了可用于标识自身的常量。通过编写一个支持每种编译器的代码段，可以创建可移植的程序。程序清单 D.7 演示了这种做法。

程序清单 D.7　　一个可移植的、包含编辑器特定元素的程序

```
1:  /*=================================================*
2:   * Program: listD07.c                               *
3:   * Purpose: This program demonstrates using defined *
4:   *          constants for creating a portable program *
5:   * Note:    This program gets different results with *
6:   *          different compilers.                     *
7:   *=================================================*/
8:  #include <stdio.h>
9:  #ifdef _WINDOWS
10:
11:  #define STRING "DOING A WINDOWS PROGRAM!\n"
12:
13:  #else
14:
15:  #define STRING "NOT DOING A WINDOWS PROGRAM\n"
16:
17:  #endif
18:
19:  int main(void)
20:  {
21:    printf( "\n\n") ;
22:    printf( STRING );
23:
24:  #ifdef _MSC_VER
25:
```

www.plcworld.cn


```
26:    printf( "\n\nUsing a Microsoft compiler!" );
27:    printf( "\n   Your Compiler version is %s\n", _MSC_VER );
28:
29: #endif
30:
31: #ifdef __TURBOC__
32:
33:    printf( "\n\nUsing the Turbo C compiler!" );
34:    printf( "\n   Your compiler version is %x\n", __TURBOC__ );
35:
36: #endif
37:
38: #ifdef __BORLANDC__
39:
40:    printf( "\n\nUsing a Borland compiler!\n" );
41:
42: #endif
43:
44:    return(0);
45: }
```

使用 Turbo C for DOS 3.0 编译器来编译并运行该程序时，输出如下：

```
NOT DOING A WINDOWS PROGRAM

Using the Turbo C compiler!
      Your compiler version is 300
```

在 DOS 下，使用 Borland C++编译器来编译并运行该程序时，输出如下：

```
NOT DOING A WINDOWS PROGRAM

Using a Borland compiler!
```

在 DOS 下，使用 Microsoft 编译器来编译并运行该程序时，输出如下：

```
NOT DOING A WINDOWS PROGRAM

Using a Microsoft compiler!
      Your compiler version is >>
```

分析：该程序使用预定义的常量来获得有关编译器的信息。第 9 行使用了预处理器编译指令#ifdef。该编译指令检查后面的常量是否被定义。如果是，则执行预处理器编译指令#ifdef 和#endif 之间的语句。第 9 行判断_WINDOWS 是否已被定义。然后根据情况，将 STRING 定义为一条相应的消息。第 22 行打印该字符串，指出程序是否被编译为一个 Windows 程序。

第 24 行检查_MSC_VER 是否被定义。_MSC_VER 是一个常量，其值为 Microsoft 编译器的版本号。如果使用的不是 Microsoft 编译器，则该常量未被定义；否则该常量的值为 Microsoft 编译器的版本号。第 26 行指出使用的是一个 Microsoft 编译器，然后第 27 行打印该编译器的版本号。

第 31～36 行和第 38～42 行的原理与此类似。它们分别检查使用的编译器是否为 Borland Turbo C 和 Borland 的专业编译器，并根据检查结果打印相应的消息。

正如您看到的，该程序通过检查预定义的常量来确定使用的编译器。不管使用的是哪种编译器，程序的目标都相同——打印一条消息，指出使用的是哪种编译器。如果您知道程序将被移植到哪种系统中，则可以在代码中使用编译器特定的命令。在这种情况下，应针对每种编译器提供相应的代码。

D.8.4　ANSI 标准头文件

ANSI 标准规定了几个可包含的头文件。知道哪些头文件属于 ANSI 标准是很有帮助的，因为创建可移植

www.plcworld.cn

的程序时，可以使用这些文件。附录 C 列出了 ANSI 标准中的头文件及其包含的函数。

D.9 总 结

本附录介绍的内容很多，这些内容都是与移植性相关的。C 语言是可移植性最强的语言之一。移植性不是偶然出现的。制定 ANSI 标准的目的就是为了确保 C 语言程序能够从一种编译器和系统中被移植到另一种编译器和系统中。编写可移植的代码时，有多个方面需要考虑，包括变量名的大小写、字符比较、结构和共用体的使用以及预处理器编译指令和预处理器常量的使用等。本附录的最后介绍了如何在可移植的程序中使用编译器特定的元素。

D.10 问与答

问：如何编写可移植的图形程序？

答：ANSI 没有针对图形编程制定任何标准。与其他编程领域相比，图形编程与机器的关系更为密切，因此编写可移植的图形程序有些困难。

如果您决定编写图形函数，一种可选的方法是将图形例程封装到一个函数中。这样，当您改变平台时，可以使用其他内容替换该函数。

问：始终都应考虑移植性吗？

答：不。并非总需要考虑移植性。您编写的有些程序只在您的系统上运行。另外，有些程序不会被移植到其他计算机系统中。因此可以使用不可移植的函数，如 system()，而这些函数是不能用于可移植的程序中的。

问：如果使用//而不是/*和*/来进行注释，这些注释是可移植的吗？

答：根据 ISO/IEC 9899:1999 标准（最新的 ANSI 标准），可以使用//来进行注释。旧的 ANSI 标准不支持这种注释，但很多编译器都支持。

D.11 作 业

下面的小测验帮助您巩固所学的知识，练习则让您实际应用所学的知识。

D.11.1 小测验

1. 效率和可维护性哪一方面更重要？
2. 字母 a 对应的数值是多少？
3. 在您的系统中，unsigned char 变量的最大取值是多少？
4. ANSI 代表的是什么？
5. 在 C 语言程序中，下列变量名是否有效：

```
int lastname,
LASTNAME,
LastName,
Lastname;
```

6. 函数 isalpha()有何用途？
7. 函数 isdigit()有何用途？
8. 为何要使用诸如 isalpha()和 isdigit()等函数？

www.plcworld.cn

9. 将结构写入磁盘时，需要考虑移植性吗？

10. 可以像下面这样，在 printf()语句中使用__TIME__来打印当前时间吗？

```
printf( "The Current Time is: %s", __TIME__ );
```

D.11.2　练习

1. 排错：下面的函数是否有错？如果有，错在哪里？

```
void Print_error( char *msg )
{
      static int ctr = 0,
                 CTR = 0;
      printf("\n" );
      for( ctr = 0; ctr < 60; ctr++ )
      {
        printf("**");
      }
      printf( "\nError %d, %s - %d: %s.\n", CTR,
            __FILE__, __LINE__, msg );
      for( ctr = 0; ctr < 60; ctr++ )
      {
        printf("**");
      }
}
```

2. 编写一个函数，检查字母是否是元音。

3. 编写一个函数，它接受一个 char 参数。如果该参数不是一个字母，则函数返回 0；如果参数是一个大写字母，则返回 1；如果是一个小写字母，则返回 2。应使该函数的可移植性尽可能高。

4. 选做题：研究您的编译器。确定哪个标记用于忽略变量名的大小写，哪个用于启用字对齐，哪个用于确保 ANSI 兼容性。

5. 下面的代码是可移植的吗？

```
void list_a_file( char *file_name )
{
      system("TYPE " file_name );
}
```

6. 下面的代码是可移植的吗？

```
int to_upper( int x )
{
      if( x >= 'a' && x <= 'z')
      {
            toupper( x );
      }
      return( x );
}
```

www.plcworld.cn

附录 E　常用的 C 语言函数

本附录列出了大多数 C 语言编译器都提供了的头文件中包含的函数原型，其中后面带星号的函数是本书介绍过的。

这些函数是按字母顺序列出的。除了函数名外，还指出了该函数所在的头文件及其原型。注意，在头文件中，原型的表示方法与本书不同——对于函数接受的每个参数，原型中只给出了类型，而没有参数名。例如：

```
int func1(int, int *);
int func1(int x, int *y);
```

上述两个声明都指定了两个参数——第一个参数的类型为 int，第二个为 int 指针。在编译器看来，这两个声明是等价的。

表 E.1　常用的 C 语言函数（按字母顺序排列）

函　数	头 文 件	函数原型
abort*	stdlib.h	void abort(void);
abs	stdlib.h	int abs(int);
acos*	math.h	double acos(double);
asctime*	time.h	char *asctime(const struct tm *);
asin*	math.h	double asin(double);
assert*	assert.h	void assert(int);
atan*	math.h	double atan(double);
atan2*	math.h	double atan2(double, double);
atexit*	stdlib.h	int atexit(void (*)(void));
atof*	stdlib.h	double atof(const char *);
atof*	math.h	double atof(const char *);
atoi*	stdlib.h	int atoi(const char *);
atol*	stdlib.h	long atol(const char *);
bsearch*	stdlib.h	void *bsearch(const void *, const void *, size_t, size_t, int(*) (const void *, const void *));
calloc*	stdlib.h	void *calloc(size_t, size_t);
ceil*	math.h	double ceil(double);
clearerr	stdio.h	void clearerr(FILE *);
clock*	time.h	clock_t clock(void);
cos*	math.h	double cos(double);
cosh*	math.h	double cosh(double);
ctime*	time.h	char *ctime(const time_t *);
difftime	time.h	double difftime(time_t, time_t);
div	stdlib.h	div_t div(int, int);

www.plcworld.cn

续表

函　数	头 文 件	函数原型
exit*	stdlib.h	void exit(int);
exp*	math.h	double exp(double);
fabs*	math.h	double fabs(double);
fclose*	stdio.h	int fclose(FILE *);
fcloseall*	stdio.h	int fcloseall(void);
feof*	stdio.h	int feof(FILE *);
fflush*	stdio.h	int fflush(FILE *);
fgetc*	stdio.h	int fgetc(FILE *);
fgetpos	stdio.h	int fgetpos(FILE *, fpos_t *);
fgets*	stdio.h	char *fgets(char *, int, FILE *);
floor*	math.h	double floor(double);
flushall*	stdio.h	int flushall(void);
fmod*	math.h	double fmod(double, double);
fopen*	stdio.h	FILE *fopen(const char *, const char *);
fprintf*	stdio.h	int fprintf(FILE *, const char *, ...);
fputc*	stdio.h	int fputc(int, FILE *);
fputs*	stdio.h	int fputs(const char *, FILE *);
fread*	stdio.h	size_t fread(void *, size_t, size_t, FILE *);
free*	stdlib.h	void free(void *);
freopen	stdio.h	FILE *freopen(const char *, const char *, FILE*);
frexp*	math.h	double frexp(double, int *);
fscanf*	stdio.h	int fscanf(FILE *, const char *, ...);
fseek*	stdio.h	int fseek(FILE *, long, int);
fsetpos	stdio.h	int fsetpos(FILE *, const fpos_t *);
ftell*	stdio.h	long ftell(FILE *);
fwrite*	stdio.h	size_t fwrite(const void *, size_t, size_t, FILE *);
getc*	stdio.h	int getc(FILE *);
getch*	stdio.h	int getch(void);
getchar*	stdio.h	int getchar(void);
getche*	stdio.h	int getche(void);
getenv	stdlib.h	char *getenv(const char *);
gets*	stdio.h	char *gets(char *);
gmtime	time.h	struct tm *gmtime(const time_t *);
isalnum*	ctype.h	int isalnum(int);
isalpha*	ctype.h	int isalpha(int);
isascii*	ctype.h	int isascii(int);
iscntrl*	ctype.h	int iscntrl(int);
isdigit*	ctype.h	int isdigit(int);
isgraph*	ctype.h	int isgraph(int);
islower*	ctype.h	int islower(int);
isprint*	ctype.h	int isprint(int);

www.plcworld.cn

续表

函　数	头 文 件	函数原型
ispunct*	ctype.h	int ispunct(int);
isspace*	ctype.h	int isspace(int);
isupper*	ctype.h	int isupper(int);
isxdigit*	ctype.h	int isxdigit(int);
labs	stdlib.h	long int labs(long int);
ldexp	math.h	double ldexp(double, int);
ldiv	stdlib.h	ldiv_t div(long int, long int);
localtime*	time.h	struct tm *localtime(const time_t *);
log*	math.h	double log(double);
log10*	math.h	double log10(double);
malloc*	stdlib.h	void *malloc(size_t);
mblen	stdlib.h	int mblen(const char *, size_t);
mbstowcs	stdlib.h	size_t mbstowcs(wchar_t *, const char *, size_t);
mbtowc	stdlib.h	int mbtowc(wchar_t *, const char *, size_t);
memchr	string.h	void *memchr(const void *, int, size_t);
memcmp	string.h	int memcmp(const void *, const void *, size_t);
memcpy	string.h	void *memcpy(void *, const void *, size_t);
memmove	string.h	void *memmove(void *, const void*, size_t);
memset	string.h	void *memset(void *, int, size_t);
mktime*	time.h	time_t mktime(struct tm *);
modf	math.h	double modf(double, double *);
perror*	stdio.h	void perror(const char *);
pow*	math.h	double pow(double, double);
printf*	stdio.h	int printf(const char *, ...);
putc*	stdio.h	int putc(int, FILE *);
putchar*	stdio.h	int putchar(int);
puts*	stdio.h	int puts(const char *);
qsort*	stdlib.h	void qsort(void*, size_t, size_t, int (*)(const void*, const void *));
rand	stdlib.h	int rand(void);
realloc*	stdlib.h	void *realloc(void *, size_t);
remove*	stdio.h	int remove(const char *);
rename*	stdio.h	int rename(const char *, const char *);
rewind*	stdio.h	void rewind(FILE *);
scanf*	stdio.h	int scanf(const char *, ...);
setbuf	stdio.h	void setbuf(FILE *, char *);
setvbuf	stdio.h	int setvbuf(FILE *, char *, int, size_t);
sin*	math.h	double sin(double);
sinh*	math.h	double sinh(double);
sleep*	time.h	void sleep(time_t);
sprintf	stdio.h	int sprintf(char *, const char *, ...);
sqrt*	math.h	double sqrt(double);

www.plcworld.cn

续表

函 数	头 文 件	函数原型
srand	stdlib.h	void srand(unsigned);
sscanf	stdio.h	int sscanf(const char *, const char *, ...);
strcat*	string.h	char *strcat(char *,const char *);
strchr*	string.h	char *strchr(const char *, int);
strcmp*	string.h	int strcmp(const char *, const char *);
strcmpl*	string.h	int strcmpl(const char *, const char *);
strcpy*	string.h	char *strcpy(char *, const char *);
strcspn*	string.h	size_t strcspn(const char *, const char *);
strdup*	string.h	char *strdup(const char *);
strerror	string.h	char *strerror(int);
strftime*	time.h	size_t strftime(char *, size_t, const char *, const struct tm *);
strlen*	string.h	size_t strlen(const char *);
strlwr*	string.h	char *strlwr(char *);
strncat*	string.h	char *strncat(char *, const char *, size_t);
strncmp*	string.h	int strncmp(const char *, const char *, size_t);
strncpy*	string.h	char *strncpy(char *, const char *, size_t);
strnset*	string.h	char *strnset(char *, int, size_t);
strpbrk*	string.h	char *strpbrk(const char *, const char *);
strrchr*	string.h	char *strrchr(const char *, int);
strspn*	string.h	size_t strspn(const char *, const char *);
strstr*	string.h	char *strstr(const char *, const char *);
strtod	stdlib.h	double strtod(const char *, char **);
strtok	string.h	char *strtok(char *, const char*);
strtol	stdlib.h	long strtol(const char *, char **, int);
strtoul	stdlib.h	unsigned long strtoul(const char*, char **, int);
strupr*	string.h	char *strupr(char *);
system*	stdlib.h	int system(const char *);
tan*	math.h	double tan(double);
tanh*	math.h	double tanh(double);
time*	time.h	time_t time(time_t *);
tmpfile	stdio.h	FILE *tmpfile(void);
tmpnam*	stdio.h	char *tmpnam(char *);
tolower	ctype.h	int tolower(int);
toupper	ctype.h	int toupper(int);
ungetc*	stdio.h	int ungetc(int, FILE *);
va_arg*	stdarg.h	(type) va_arg(va_list, (type));
va_end*	stdarg.h	void va_end(va_list);
va_start*	stdarg.h	void va_start(va_list, lastfix);
vfprintf	stdio.h	int vfprintf(FILE *, constchar *, ...);
vprintf	stdio.h	int vprintf(FILE*, constchar *, ...);
vsprintf	stdio.h	int vsprintf(char *, constchar *, ...);
wcstombs	stdlib.h	size_t wcstombs(char *, const wchar_t *, size_t);
wctomb	stdlib.h	int wctomb(char *, wchar_t);

www.plcworld.cn

附录 F　作业答案

本附录提供每章最后一节中的小测验和练习的答案。注意，很多练习有多种解决方案。大部分情况下，这里只给出了其中的一种；但有时候，还提供了帮助您完成练习的其他信息。

第 1 天课程的答案

小测验

1. C 语言功能强大、流行、可移植。
2. 编译器将 C 语言源代码转换为计算机能够理解的机器语言指令。
3. 编辑、编译、链接和测试。
4. 这个问题的答案随您使用的编译器而异，请参阅编译器的用户手册。
5. 这个问题的答案随您使用的编译器而异，请参阅编译器的用户手册。
6. 合适的 C 语言源代码文件扩展名为.C（或.c）。



注意：C++使用扩展名.CPP。可以将 C 程序的扩展名设置为.CPP，但扩展名.C 更合适。

7. FILENAME.TXT 将通过编译，但扩展名.C 比.TXT 更合适。
8. 应该对源代码进行修改，以更正错误。然后重新编译和链接，并再次运行该程序，看您所做的修改是否解决了问题。
9. 机器语言由计算机能够理解的二进制指令组成。由于计算机无法理解 C 语言源代码，因此使用编译器将源代码转换为机器代码——目标代码。
10. 链接程序将程序的目标代码和函数库中的目标代码组合起来，创建一个可执行文件。

练习

1. 查看目标文件时，会看到许多怪字符和莫名其妙的东西，其中混杂着源代码片段。
2. 该程序计算机圆的面积。它提示用户输入半径，然后显示面积。
3. 该程序打印由字符 X 组成的 10×10 方块。第 6 天的课程中有一个与此类似的程序。
4. 该程序将导致编译错误，错误消息与下面类似：

```
Error: ch1ex4.c: Declaration terminated incorrectly
```

这种错误是由第 3 行末尾的分号引起的。删除这个分号后，程序就能够正确地被编译和链接。

5. 该程序能够通过编译，但链接时会发生错误。错误消息与下面类似：

```
Error: Undefined symbol _do_it in module...
```

发生错误的原因在于，链接程序找不到名为 do_it 的函数。要修复这种错误，可将 do_it 改为 printf。

www.plcworld.cn

6. 该程序打印由笑脸字符组成的 10×10 方块，而不是字符 X 组成的 10×10 方块。

第 2 天课程的答案

小测验

1. 语句块。

2. main()函数。

3. 您使用注释来说明程序的结构和工作原理。位于/*和*/的文本都是注释，编译器将忽略它们。另外，您还可以使用单行注释。两个斜杠（//）到行尾之间的文本都是注释。

4. 函数是一个独立的代码段，它执行特定的任务，并被赋予一个名称。通过函数的名称，程序可以执行该函数中的代码。

5. 用户定义的函数是由程序员创建的；而库函数是由编译器提供的。

6. 编译指令#include 命令编译器在编译时将另一个文件中的代码添加到源代码中。

7. 注释不应嵌套。虽然有些编译器允许这样做，但有些不允许。为确保代码是可移植的，不应对注释进行嵌套。

8. 是的。注释可以任意长。/*和*/之间的所有内容都是注释。

9. 包含文件也叫头文件。

10. 包含文件是一个独立的磁盘文件，其中包含编译器使用各种函数时所需的信息。

练习

1. 请记住，对于 C 语言程序而言，只有 main()函数是必不可少的。下面的程序可能是最小的，但它毫无用处：

```
void main(void)
{
}
```

这个程序也可以写成：

```
void main(void){}
```

2.

a. 第 8、9、10、12、20、21 行是语句；

b. 唯一的变量定义位于第 18 行；

c. 唯一的函数原型（display_line()的原型）位于第 4 行；

d. 函数 display_line()的定义位于第 16～22 行；

e. 注释位于第 1、15 和 23 行。

3. 注释是/*和*/之间的所有文本，例如：

```
/* This is a comment */
/*???*/
/*
This is a
     third comment */
```

您也可以使用单行注释——//到行尾之间的文本。

4. 该程序打印所有的大写字母。阅读第 10 天的课程后，您能更好地理解这个程序。

该程序的输出如下：

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

5. 该程序计算用户输入的字符和空格数，并将结果显示到屏幕上。同样，阅读第 10 天的课程后，您能更好地理解这个程序。

www.plcworld.cn

第 3 天课程的答案

小测验

1. 整型变量用于存储整数（没有小数部分的数字）；而浮点型变量用于储存实数（有小数部分的数字）。

2. double 变量的取值范围比 float 变量大（可以存储更大和更小的值）；另外，double 变量的精度也比 float 变量高。

3.

a. char 变量的长度为 1 字节；

b. short 变量的长度不超过 int 变量；

c. int 变量的长度不超过 long 变量；

d. unsigned int 变量的长度等于 int 变量；

e. float 变量的长度不超过 double 变量。

4. 符号常量的名称提高了源代码的可读性，并使修改常量的值更容易。

5.

a. `#define MAXIMUM 100`

b. `const int MAXIMUM = 100;`

6. 字母、数字和下划线。

7. 变量和常量的名称应描述它存储的数据。变量名应小写，而常量名应大写。

8. 符号常量是表示字面常量的符号。

9. 如果 unsigned int 变量的长度为 2 字节，则它能够存储的最小值为 0；如果是有符号的，则可存储的最小值为-32768。

练习

1.

a. 由于年龄是整数，且不为负，因此建议使用 unsigned int 类型；

b. unsigned int；

c. float；

d. 如果年薪不是很高，可以使用 unsigned int；如果超过了 65535 美元，则应使用 long（如果您自信心很高，应使用 long）；

e. float（美分需要用小数表示）；

f. 由于最高分数为 100，这是一个常量。因此，可使用 const int 或#define 语句；

g. float（如果只使用整数，则可以使用 int 或 long）；

h. 值肯定为正。可以使用 int、long 或 float，参见答案 1.d。

i. double。

2. 这里一并提供了练习 2 和 3 的答案。

别忘了，变量名应描述存储的值。变量声明创建了变量。在声明时，可能初始化变量的值，也可能不初始化。可以给变量使用任何名称，但不能是 C 语言的关键字。

a. `unsigned int age;`

b. `unsigned int weight;`

c. `float radius = 3;`

d. `long annual_salary;`

www.plcworld.cn

e. float cost = 29.95;

f. const int max_grade = 100; or #define MAX_GRADE 100

g. float temperature;

h. long net_worth = -30000;

i. double star_distance;

3. 参见答案 2。

4. 其中 b、c、e、g、h、i 和 j 是合法的变量名。

注意，j 是合法的，但使用这样长的名称是不明智的（谁愿意输入这样长的名称？）大多数编译器不会使用整个名称，而只使用前 31 个字符。

下面的做法是非法的：

a. 变量名以数字开头；

b. 在变量名中使用#；

c. 在变量名中使用短线（-）。

第 4 天课程的答案

小测验

1. 这是一条赋值语句，它命令计算机将 5 和 8 相加，并将结果赋给变量 x。

2. 表达式是结果为数值的任何东西。

3. 运算符的相对优先级。

4. 第一条语句执行后，a 的值为 10，x 的值为 11；第二条语句执行后，a 和 x 的值都为 11（语句必须单独执行）。

5. 1。

6. 19。

7. (5 + 3) * 8 / (2 + 2)

8. 0。

9. 请参见本章最后的“再谈运算符优先级”一节，其中列出了 C 语言中的运算符及其优先级。

a. <的优先级高于==；

b. *的优先级比+高；

c. !=的优先级与==相同，因此从左到右进行计算；

d. >=的优先级与>相同；如果在一个语句或表达式中需要使用多个关系运算符，应使用圆括号。

10. 复合赋值运算符让您能够将双目数学运算和赋值运算组合在一起，从而提供了一种简明的表示方式。本章介绍的复合运算符有+=、-=、/=、*=和%=。

练习

1. 该程序能够运行，虽然其格式非常糟糕。该程序表明，空白不会影响程序的运行。您应使用空白来提高程序的可读性。

2. 下面是练习 1 中代码的更好的组织方式：

```
#include <stdio.h>

int x, y;

int main( void )
```

www.plcworld.cn


```
{
        printf("\nEnter two numbers ");
        scanf( "%d %d",&x,&y);
        printf("\n\n%d is bigger\n",(x>y)?x:y);
        return 0;
}
```

该程序提示用户输入两个数字，并将其存储到 x 和 y 中，然后打印较大的一个。

3. 需要做的修改如下：

```
16:    printf("\n%d   %d", a++, ++b);
17:    printf("\n%d   %d", a++, ++b);
18:    printf("\n%d   %d", a++, ++b);
19:    printf("\n%d   %d", a++, ++b);
20:    printf("\n%d   %d", a++, ++b);
```

4. 下面的代码片段是可能的答案之一。它检查 x 是否大于等于 1，且小于等于 20。如果这两个条件都满足，则将 x 赋给 y；否则，不将 x 赋给 y，因此 y 的值保持不变。

```
if ((x >= 1) && (x <= 20))
        y = x;
```

5. 代码如下：

```
y = ((x >= 1) && (x <= 20)) ? x : y;
```

同样，如果条件为 TRUE，则将 x 赋给 y；否则将 y 赋给 y，这相当于没有进行赋值。

6. 代码如下：

```
if (x < 1 && x > 10 )
        statement;
```

7.

a. 7

b. 0

c. 9

d. 1 (true)

e. 5

8.

a. TRUE;

b. FALSE;

c. TRUE。其中只有一个等号，因此 if 语句进行赋值，而不是进行比较。

d. TRUE。

9. 下面是解决方案之一：

```
if( age < 21 )
        printf( "You are not an adult" );
else if( age >= 65 )
        printf( "You are a senior citizen!");
    else
        printf( "You are an adult" );
```

10. 这个程序有 4 个问题。首先是第 3 行，赋值语句应以分号（而不是逗号）结尾。第二个问题是，第 6 行的 if 语句以分号结尾。第三个问题比较常见：在 if 语句中使用赋值运算符（=），而不是关系运算符（==）。最后一个问题是第 8 行的 otherwise，应将其改为 else。更正后的代码如下：

```
/* a program with problems... */
#include <stdio.h>
int x = 1;
int main( void )
```

www.plcworld.cn


```
{
    if( x == 1)
       printf(" x equals 1" );
    else
       printf(" x does not equal 1");
    return 0;
}
```

第 5 天课程的答案

小测验

1. 是的（这个问题有些恶作剧，但要成为一名优秀的 C 语言程序员，您必须做出肯定的回答）！

2. 结构化编程将复杂的任务划分为多个简单的任务，这些任务更容易处理。

3. 将程序划分为多个简单的任务后，便可以编写完成每种任务的函数。

4. 函数定义的第一行为函数头，其中包含函数的名称、返回类型和参数列表。

5. 函数可以返回一个值或不返回任何值。返回值可以是任何类型。第 18 天的课程介绍了如何从函数获取多个值。

6. 不返回任何值的函数的类型为 void。

7. 函数定义是完整的函数，其中包括函数头和代码。函数定义决定了函数被执行时将执行哪些操作。函数原型只有一行，它与函数头相同，但以分号结尾。原型将函数的名称、返回类型和参数列表告诉编译器。

8. 局部变量是在函数中声明的。

9. 局部变量独立于程序中的其他变量。

10. 程序的第一个函数应为 main()。

练习

1. float do_it(char a, char b, char c)

在后面加上分号便可以得到函数原型。作为函数头时，后面应是用花括号括起的函数代码。

2. void print_a_number(int a_number)

这是一个 void 函数。与练习 1 一样，要创建原型，只需在后面加上分号即可。在程序中，函数头后面应包含函数的代码。

3.

a. int

b. long

4. 有两个问题。首先，print_msg()被声明为 void 类型，但却返回一个值。应将 return 语句删除。其次是第 5 行，它调用 print_msg()时传递了一个参数（一个字符串）。函数原型指出，该函数的参数列表为 void，因此不应给它传递任何参数。更正后的代码如下：

```
#include <stdio.h>
void print_msg (void);
int main( void )
{
        print_msg();
        return 0;
}
void print_msg(void)
{
```

www.plcworld.cn


```
        puts( "This is a message to print" );
}
```

5. 函数头不应以分号结尾。

6. 只需修改函数 large_of()即可：

```
21: int larger_of( int a, int b)
22: {
23:     int save;
24:
25:     if (a > b)
26:         save = a;
27:     else
28:         save = b;
29:
30:     return save;
31: }
```

7. 下面的代码假设这两个值为 int 类型，并返回一个 int 值：

```
int product( int x, int y )
{
    return (x * y);
}
```

8. 下面的代码检查传递第二个值是否为 0，因为除以 0 将出错。决不要假设传递的值是正确的。

```
int divide_em( int a, int b )
{
        int answer = 0;

        if( b == 0 )
          answer = 0;
        else
          answer = a/b;

        return answer;
}
```

9. 虽然下面的代码使用的是 main()函数，但也可以使用任何函数。第 9～11 行调用了两个函数；第 13～16 行打印值。要运行该程序，必须在最后加上练习 7 和 8 中的代码。

```
1:  #include <stdio.h>
2:
3:  int main( void )
4:  {
5:     int number1 = 10,
6:         number2 = 5;
7:     int x, y, z;
8:
9:     x = product( number1, number2 );
10:    y = divide_em( number1, number2 );
11:    z = divide_em( number1, 0 );
12:
13:     printf( "\nnumber1 is %d and number2 is %d", number1, number2 );
14:     printf( "\nnumber1 * number2 is %d", x );
15:     printf( "\nnumber1 / number2 is %d", y );
16:     printf( "\nnumber1 / 0 is %d", z );
17:
```

www.plcworld.cn


```
18:     return 0;
19: }
```

10. 代码如下：

```
/* Averages five float values entered by the user. */

#include <stdio.h>

float v, w, x, y, z, answer;

float average(float a, float b, float c, float d, float e);

int main( void )
{
        puts("Enter five numbers:");
        scanf("%f%f%f%f%f", &v, &w, &x, &y, &z);

        answer = average(v, w, x, y, z);

        printf("The average is %f.\n", answer);

        return 0;
}

float average( float a, float b, float c, float d, float e)
{
        return ((a+b+c+d+e)/5);
}
```

11. 下面的代码使用的是int变量，因此使用的值最大不能超过9。要使用更大的值，应将变量的类型声明为long。

```
/* this is a program with a recursive function */

#include <stdio.h>

int three_powered( int power );
int main( void )
{
        int a = 4;
        int b = 9;

        printf( "\n3 to the power of %d is %d", a,
        three_powered(a) );
        printf( "\n3 to the power of %d is %d\n", b,
        three_powered(b) );

        return 0;
}

int three_powered( int power )
{

        if ( power < 1 )
              return( 1 );
        else
```

www.plcworld.cn


```
        return( 3 * three_powered( power - 1 ));
    }
```

第 6 天课程的答案

小测验

1. 在 C 语言中，数组的第一个索引值为 0。
2. for 语句中包含初始化和递增表达式。
3. do...while 循环的最后为 while 语句，这种循环至少执行一次。
4. 是的，while 循环也可以完成 for 语句的工作，但您需要另外做两项工作。必须在 while 循环之前对变量进行初始化，并在 while 循环中对变量进行递增。
5. 循环不能彼此重叠。被嵌套的循环必须完全位于外部循环中。
6. 是的，可以将 while 语句嵌套到 do...while 循环中。可以在任何命令中嵌套其他任何命令。
7. for 循环的 4 部分是初始化部分、条件部分、递增部分和语句。
8. while 循环的两个部分是条件和语句。
9. do...while 循环的两个部分是条件和语句。

练习

1. long array[50];
2. 在下面的答案中，第 50 个元素的索引为 49。别忘了，数组的索引从 0 开始。

```
array[49] = 123.456;
```

3. 执行完语句后，x 的值为 100。
4. 语句执行完毕后，ctr 的值为 11（ctr 从 2 开始，每次递增 3，直到大于等于 10 为止）。
5. 内面的循环打印 5 个 X，外面的循环执行内面的循环 10 次。因此，总共打印 50 个 X。
6. 代码如下：

```
int x;
for( x = 1; x <= 100; x += 3) ;
```

7. 代码如下：

```
int x = 1;
while( x <= 100 )
        x += 3;
```

8. 代码如下：

```
int ctr = 1;
do
{
        ctr += 3;
} while( ctr < 100 );
```

9. 该程序永远不会终止。该程序首先将 record 初始化为 0，然后，while 循环检查 record 是否小于 100。而 0 确实小于 100，因此执行循环，打印两个值。然后，循环再次检查这一条件，条件再次成立，因此再次执行循环。这一过程将不断地重复下去。在循环体内，应递增 record 的值。应在第二个 printf()函数调用的后面加上下面的代码行：

```
record++;
```

10. 循环时，经常使用定义的常量：第 2 和第 3 周课程中有一些与此类似的例子。该代码片段存在的问题很简单，for 语句不能以分号结尾。这是一种常见的错误。

www.plcworld.cn

第 7 天课程的答案

小测验

1. puts()和 printf()之间的差别有两点：
 printf()接受可变数目的参数；
 puts()打印完字符串后自动换行。
2. 使用 printf()时，应包含头文件 stdio.h。
3.
 a. \\打印一个反斜杠；
 b. \b 退格；
 c. \n 换行；
 d. \t 打印制表符；
 e. \a 振铃。
4.
 a. %a 用于字符串；
 b. %d 用于有符号的十进制整数；
 c. %f 用于十进制浮点数。
5.
 a. b 打印字符 b；
 b. \b 退格；
 c. \后面为转义字符（参见表 7.1）；
 d. \\打印一个反斜杠。

练习

1. puts()会自动换行，而 printf()不会。代码如下：

```
printf( "\n" );
puts( " " );
```

2. 代码如下：

```
char c1, c2;
int d1;
scanf( "%c %ud %c", &c1, &d1, &c2 );
```

3. 您的答案可能与此不同：

```
#include <stdio.h>
int x;

int main( void )
{
    puts( "Enter an integer value" );
    scanf( "%d", &x );

    printf( "\nThe value entered is %d\n", x );

    return 0;
```

www.plcworld.cn


```
}
```

4. 程序经常只接受特定的值。该练习的解决方式之一是：

```
   #include <stdio.h>int x;
int main( void )
{
        puts( "Enter an even integer value" );
        scanf( "%d", &x );
        while( x % 2 != 0)
        {
              printf( "\n%d is not even, Please enter an even \
              number: ", x );
              scanf( "%d", &x );
        }
        printf( "\nThe value entered is %d\n", x );
        return 0;
}
```

5. 代码如下：

```
#include <stdio.h>
int array[6], x, number;

int main( void )
{
        /* loop 6 times or until the last entered element is 99 */
        for( x = 0; x < 6 && number != 99; x++ )
        {
           puts( "Enter an even integer value, or 99 to quit" );
           scanf( "%d", &number );
           while( number % 2 == 1 && number != 99)
           {
               printf( "\n%d is not even, Please enter an even \
                    number: ", number);
               scanf( "%d", &number );
           }
           array[x] = number;
        }
        /* now print them out... */
        for( x = 0; x < 6 && array[x] != 99; x++ )
        {
           printf( "\nThe value entered is %d", array[x] );
        }

        return 0;
}
```

6. 前一个解决方案已经是可执行程序，只需修改最后一个 printf()。要将打印的值用制表符分开，可将 printf() 语句修改成如下所示：

```
printf( "%d\t", array[x]);
```

7. 不能在引号中包含引号。要打印引号，必须使用转义字符\"。另外，必须在第一行的行尾加上一个反斜杠，以便将代码行扩展到下一行。正确的代码如下：

```
printf( "Jack said, \"Peter Piper picked a peck of pickled \
peppers.\"");
```

www.plcworld.cn

8. 该程序清单有三个错误。首先，printf()语句中没有使用引号；其次，在scanf()中，变量answer前面没有地址运算符；最后，scanf()语句中应使用%d，而不是%f，因为answer是一个int变量，而不是float变量。更正后的代码如下：

```
int get_1_or_2( void )
{
        int answer = 0;

        while( answer < 1 || answer > 2 )
        {
            printf("Enter 1 for Yes, 2 for No ");     /* corrected */

            scanf( "%d", &answer );                   /* corrected */
        }
        return answer;
}
```

9. 下面是程序清单7.1中的print_report()函数的代码：

```
void print_report( void )
{
        printf( "\nSAMPLE REPORT" );
        printf( "\n\nSequence\tMeaning" );
        printf( "\n========\t=======" );
        printf( "\n\\a\t\tBell (alert)" );
        printf( "\n\\b\t\tBackspace" );
        printf( "\n\\f\t\tForm feed" );
        printf( "\n\\n\t\tNew line" );
        printf( "\n\\r\t\tCarriage Return" );
        printf( "\n\\t\t\tHorizontal tab" );
        printf( "\n\\v\t\tVertical tab" );
        printf( "\n\\\\\t\tBackslash" );
        printf( "\n\\?\t\tQuestion mark" );
        printf( "\n\\'\t\tSingle quote");
        printf( "\n\\"\t\tDouble quote" );
        printf( "\n...\t\t...");
}
```

10. 代码如下：

```
/* Inputs two floating point values and */
/* displays their product. */

#include <stdio.h>

float x, y;

int main( void )
{
    puts("Enter two values: ");
    scanf("%f %f", &x, &y);
    printf("\nThe product is %f\n", x * y);
    return 0;
}
```

11. 下面的程序提示用户输入10个数字，然后显示它们总和：

```
/* Input 10 integers and display their sum. */
```

www.plcworld.cn


```
#include <stdio.h>

int count, temp;
long total = 0;     /* Use type long to ensure we don't*/
/* exceed the maximum for type int. */

int main( void )
{
        for (count = 1; count <=10; count++)
        {
                printf("Enter integer # %d: ", count);
                scanf("%d", &temp);
                total += temp;
        }

        printf("\n\nThe total is %d\n", total);

        return 0;
}
```

12. 代码如下：

```
/* Inputs integers and stores them in an array, stopping */
/* when a zero is entered. Finds and displays the array's */
/* largest and smallest values */
#include <stdio.h>

#define MAX 100

int array[MAX];
int count = -1, maximum, minimum, num_entered, temp;

int main( void )
{
        puts("Enter integer values one per line.");
        puts("Enter 0 when finished.");

        /* Input the values */

        do
        {
                scanf("%d", &temp);
                array[++count] = temp;
        } while ( count < (MAX-1) && temp != 0 );

        num_entered = count;

        /* Find the largest and smallest. */
        /* First set maximum to a very small value, */
        /* and minimum to a very large value. */

        maximum = -32000;
        minimum = 32000;
```

www.plcworld.cn


```
        for (count = 0; count <= num_entered && array[count] != 0; count++)
        {
                if (array[count] > maximum)
                maximum = array[count];

                if (array[count] < minimum )
                        minimum = array[count];
        }

        printf("\nThe maximum value is %d", maximum);
        printf("\nThe minimum value is %d\n", minimum);

        return 0;
}
```

第 8 天课程的答案

小测验

1. 任何类型，但只能同时使用一种。特定数组只能是一种数据类型。
2. 0。不管数组的长度如何，其下标都从 0 开始。
3. n-1。
4. 程序能够编译并运行，但结果是不可预测的。
5. 在声明语句中，数组名后有多对方括号，每维一对。其中每对方括号一个数字，它指定了相应维的元素数目。
6. 240。计算方法为 2×3×5×8。
7. array[0][0][1][1]。
8. sizeof(xyz) / sizeof(long)。

练习

1. int one[1000], two[1000], three[1000];
2. int array[10] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
3. 解决方法有很多。一种是在声明数组时对其进行初始化：

```
int eightyeight[88] = {88,88,88,88,88,88,88,...,88};
```

然而，使用这种方法，需要将 88 个 88 放在花括号中。对于大型数组，不推荐使用这种方法进行初始化。下面是一种更好的解决方案：

```
int eightyeight[88];
int x;

for ( x = 0; x < 88; x++ )
        eightyeight[x] = 88;
```

4. 代码如下：

```
int stuff[12][10];
int sub1, sub2;

for( sub1 = 0; sub1 < 12; sub1++ )
        for( sub2 = 0; sub2 < 10; sub2++ )
```

www.plcworld.cn


```
                stuff[sub1][sub2] = 0;
```

5. 该代码片段说明了人们常犯的错误。数组为 10×3，但初始化时为 3×10。

第一个下标为 10，但 for 循环将 x 作为第一个下标，而 x 只被递增三次。第二个下标被声明为 3，第二个 for 循环使用 y 作为第二个下标，而 y 递增了 10 次。这可能导致不可预测的结果。可以采用两种方式之一来修复这种问题。第一种方式是，将初始化语句中 x 和 y 的位置对换：

```
int x, y;
int array[10][3];
int main( void )
{
      for ( x = 0; x < 3; x++ )
            for ( y = 0; y < 10; y++ )
                  array[y][x] = 0;        /* changed */

      return 0;
}
```

第二种方式是，对换 for 循环中的值（推荐采用这种方式）：

```
int x, y;
int array[10][3];
int main( void )
{
      for ( x = 0; x < 10; x++ )      /* changed */
            for ( y = 0; y < 3; y++ )   /* changed */
                  array[x][y] = 0;

      return 0;
}
```

6. 这种错误很容易排除。该程序初始化数组边界外的一个元素。如果数组包含 10 个元素，则下标为 0～9。该程序初始化下标为 1～10 的元素。不能初始化 array[10]，因为它不存在。应将 for 语句修改为下面的任何一个：

```
for( x = 1; x <=9; x++ )  /* initializes 9 of the 10 elements */
for( x = 0; x <= 9; x++ )
```

注意，x<=9 与 x<10 等效。它们都可行，但 x<10 更常用。

7. 下面是解决方案之一：

```
/* Using two-dimensional arrays and rand() */

#include <stdio.h>
#include <stdlib.h>

/* Declare the array */
int array[5][4];
int a, b;

int main( void )
{
      for ( a = 0; a < 5; a++ )
      {
            for ( b = 0; b < 4; b++ )
            {
                  array[a][b] = rand();
            }
      }
```

www.plcworld.cn


```
        /* Now print the array elements */

        for ( a = 0; a < 5; a++ )
        {
                for ( b = 0; b < 4; b++ )
                {
                        printf( "%d\t", array[a][b] );
                }
                printf( "\n" );    /* go to a new line */
        }

        return 0;
}
```

8. 下面是解决方案之一：

```
/* random.c: using a single-dimensional array */

#include <stdio.h>
#include <stdlib.h>
/* Declare a single-dimensional array with 1000 elements */

int random[1000];
int a, b, c;
long total = 0;

int main( void )
{
      /* Fill the array with random numbers. The C library */
      /* function rand() returns a random number. Use one */
      /* for loop for each array subscript. */

      for (a = 0; a < 1000; a++)
      {
              random[a] = rand();
              total += random[a];
      }
      printf("\n\nAverage is: %ld\n",total/1000);
      /* Now display the array elements 10 at a time */
      for (a = 0; a < 1000; a++)
      {
              printf("\nrandom[%d] = ", a);
              printf("%d", random[a]);

              if ( a % 10 == 0 && a > 0 )
              {
                    printf("\nPress Enter to continue, CTRL-C to quit.");
                    getchar();
              }
      }
      return 0;
}       /* end of main() */
```

9. 下面是两种解决方案。前者在声明数组时对其进行初始化；后者使用 for 循环进行初始化。

www.plcworld.cn

解决方案 1：

```
#include <stdio.h>

/* Declare a single-dimensional array */

int elements[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int idx;

int main( void )
{
      for (idx = 0; idx < 10; idx++)
      {
            printf( "\nelements[%d] = %d ", idx, elements[idx] );
      }
      return 0;
}       /* end of main() */
```

解决方案 2：

```
#include <stdio.h>

/* Declare a single-dimensional array */

int elements[10];
int idx;

int main( void )
{
      for (idx = 0; idx < 10; idx++)
            elements[idx] = idx ;

      for (idx = 0; idx < 10; idx++)
            printf( "\nelements[%d] = %d ", idx, elements[idx] );

      return 0;
}
```

10. 下面是解决方案之一：

```
#include <stdio.h>

/* Declare a single-dimensional array */

int elements[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int new_array[10];
int idx;

int main( void )
{
      for (idx = 0; idx < 10; idx++)
      {
            new_array[idx] = elements[idx] + 10 ;
      }

      for (idx = 0; idx < 10; idx++)
```

www.plcworld.cn


```
        {
                printf( "\nelements[%d] = %d \tnew_array[%d] = %d",
                        idx, elements[idx], idx, new_array[idx] );
        }
        return 0;
}
```

第 9 天课程的答案

小测验

1. 地址运算符为&。
2. 使用间接运算符*。在指针名前加上*时，引用的是指针指向的变量。
3. 指针是一个变量，其值为另一个变量的地址。
4. 解除引用指的是使用指向变量的指针来存储该变量的值。
5. 顺序存储的，索引小的数组元素的地址值也小。
6. &data[0]和 data。
7. 一种方式是，将数组的长度作为参数传递给函数；另一种方式是，在数组中存储一个特殊的值（如 NULL），来标记数组的末尾。
8. 赋值、解除引用、取地址、递增、相减和比较。
9. 将指针相减，得到它们之间的元素数目。这里的答案为 1，它与元素的长度无关。
10. 答案仍为 1。

练习

1. char *char_ptr;
2. 下面的代码声明了 int 指针，然后将 cost 的地址赋给它：

```
int *p_cost;
p_cost = &cost;
```

3. 直接存取：cost = 100;

 间接存取：*p_cost = 100;

4. `printf( "Pointer value: %d, points at value: %d", p_cost, *p_cost);`
5. `float *variable = &radius;`
6. 代码如下：

```
data[2] = 100;
*(data + 2) = 100;
```

7. 下述代码也包含练习 8 的答案：

```
#include <stdio.h>

#define MAX1 5
#define MAX2 8

int array1[MAX1] = { 1, 2, 3, 4, 5 };
int array2[MAX2] = { 1, 2, 3, 4, 5, 6, 7, 8 };
int total;

int sumarrays(int x1[], int len_x1, int x2[], int len_x2);
```

www.plcworld.cn


```
int main( void )
{
      total = sumarrays(array1, MAX1, array2, MAX2);
      printf("The total is %d\n", total);

      return 0;
}

int sumarrays(int x1[], int len_x1, int x2[], int len_x2)
{

      int total = 0, count = 0;
      for (count = 0; count < len_x1; count++)
            total += x1[count];

      for (count = 0; count < len_x2; count++)
            total += x2[count];

      return total;
}
```

8. 参见练习 7 的答案。

9. 下面是一种解决方案：

```
#include <stdio.h>

#define SIZE 10

/* function prototypes */
void addarrays( int [], int []);

int main( void )
{
      int a[SIZE] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
      int b[SIZE] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};

      addarrays(a, b);

      return 0;
}

void addarrays( int first[], int second[])
{
      int total[SIZE];
      int *ptr_total = &total[0];
      int ctr = 0;

      for (ctr = 0; ctr < SIZE; ctr ++ )
      {
            total[ctr] = first[ctr] + second[ctr];
            printf("%d + %d = %d\n", first[ctr], second[ctr], total[ctr]);
      }
}
```

www.plcworld.cn

第 10 天课程的答案

小测验

1. 在 ASCII 字符集中，值的范围为 0~255。其中 0~127 是标准 ASCII 字符集；128~255 是扩展 ASCII 字符集。
2. 字符的 ASCII 码。
3. 字符串是以空字符结尾的字符序列。
4. 用双引号括起的一个或多个字符。
5. 用于存储字符串末尾的空字符。
6. 解释为字符对应的 ASCII 码，然后为 0（空字符的 ASCII 码）。
7.
 a. 97
 b. 65
 c. 57
 d. 32
 e. 206
 f. 6
8.
 a. l
 b. 空格
 c. c
 d. a
 e. n
 f. NUL
 g. ☺
9.

 a. 9 个字节（实际上，该变量是一个指向字符串的指针，该字符串占用 9 个字节的内存——其中 8 个用于存储字符串，另一个用于存储空字符）。

 b. 9 字节；
 c. 1 字节；
 d. 20 字节；
 e. 20 字节。
10.
 a. A
 b. A string!
 c. 0 (NULL)
 d. 这超过了字符串的末尾，因此其值是不确定的；
 e. !
 f. 字符串的第一个元素的地址。

练习

1. char letter = '$';

www.plcworld.cn

2. char array[18] = "Pointers are fun! ";
3. char *array = "Pointers are fun!";
4. 代码如下：

```
char *ptr;
ptr = malloc(81);
gets(ptr);
```

5. 下面是解决方案之一：

```
#include <stdio.h>

#define SIZE 10

/* function prototypes */
void copyarrays( int [], int []);

int main( void )
{
        int ctr=0;
        int a[SIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int b[SIZE];

        /* values before copy */
        for (ctr = 0; ctr < SIZE; ctr ++ )
        {
                printf( "a[%d] = %d, b[%d] = %d\n",
                        ctr, a[ctr], ctr, b[ctr]);
        }

        copyarrays(a, b);

        /* values after copy */
        for (ctr = 0; ctr < SIZE; ctr ++ )
        {
                printf( "a[%d] = %d, b[%d] = %d\n",
                        ctr, a[ctr], ctr, b[ctr]);
        }

        return 0;
}

void copyarrays( int orig[], int newone[])
{
        int ctr = 0;

        for (ctr = 0; ctr < SIZE; ctr ++ )
        {
                newone[ctr] = orig[ctr];
        }
}
```

6. 下面是解决方案之一：

```
#include <stdio.h>
#include <string.h>
```

www.plcworld.cn


```
/* function prototypes */
char * compare_strings( char *, char *);

int main( void )
{
        char *a = "Hello";
        char *b = "World!";
        char *longer;

        longer = compare_strings(a, b);

        printf( "The longer string is: %s\n", longer );

        return 0;
}

char * compare_strings( char * first, char * second)
{
        int x, y;

        x = strlen(first);
        y = strlen(second);

        if( x > y)
                return(first);
        else
                return(second);
}
```

7. 这是选做题。

8. a_string 被声明为一个包含 10 个字符的数组，但初始化时，使用的字符串超过 10 个字符。因此，应将 a_string 声明得更大。

9. 如果这行代码旨在初始化一个字符串，则这是错误的。应使用 char *quote 或 char quote[100]。

10. 没有错误。

11. 是的。虽然可以将一个指针赋给另一个，但不能将一个数组赋给另一个。应将赋值语句改为字符串复制函数，如 strcpy()。

第 11 天课程的答案

小测验

1. 数组中所有元素的类型都必须相同。结构的成员可以是不同的类型。

2. 结构成员运算符为句点，它用于存取结构的成员。

3. struct。

4. 结构名对应的是结构模板，它不是变量。结构实例是分配给结构的内存，可以存储数据。

5. 这些语句定义了一种结构并声明了一个名为 myaddress 的实例，然后对该实例进行初始化。结构成员 myaddress.name 被初始化为字符串“Bradley Jones”，myaddress.add1 被初始化为“RTSoftware”，myaddress.add2 被初始化为“P. O. Box 1213”，myaddress.city 被初始化为“Carmel”，myaddress.state 被初始化为“IN”，而 myaddress.zip

www.plcworld.cn

被初始化为“46032-1213”。

6. word myword;

7. 下面的语句将 ptr 指向第 二个数组元素：

```
ptr++;
```

练习

1. 代码如下：

```
struct time {
        int hours;
        int minutes;
        int seconds;
} ;
```

2. 代码如下：

```
struct data {
        int value1;
        float value2;
        float value3;

} info ;
```

3. info.value1 = 100;

4. 代码如下：

```
struct data *ptr;
ptr = &info;
```

5. 代码如下：

```
ptr->value2 = 5.5;
(*ptr).value2 = 5.5;
```

6. 代码如下：

```
struct data {
        char name[21];
        struct data *ptr;
};
```

7. 代码如下：

```
typedef struct {
        char address1[31];
        char address2[31];
        char city[11];
        char state[3];
        char zip[11];
} RECORD;
```

8. 下面的代码使用问题 5 中的值来初始化：

```
RECORD myaddress = {"RTSoftware",
                    "P.O. Box 1213",
                    "Carmel", "IN", "46032-1213" };
```

9. 该代码片段有两个错误。首先，结构应该有名称；其次，初始化 sign 的方式不对，应使用花括号将初始值括起。正确的代码如下：

```
struct zodiac {
        char zodiac_sign[21];
        int month;
} sign = {"Leo", 8};
```

www.plcworld.cn

10. 该共用体声明有一个错误。共用体只能同时存储一个变量，因此初始化共用体时，只能初始化第一个成员。正确的初始化代码如下：

```
/* setting up a union */
union data{
        char a_word[4];
        long a_number;
}generic_variable = { "WOW" };
```

第 12 天课程的答案

小测验

1. 变量的作用域指的是在程序的哪些地方可以访问它或变量在什么地方可见。

2. 局部变量只在定义它的函数中可见；全局变量在整个程序中都可见。

3. 在函数中定义的变量为局部变量；在函数外定义的变量为全局变量。

4. 动态（默认）和静态。每当函数被调用时，动态变量都被创建；而在函数结束时，动态变量将被释放。静态局部变量是永久性的，其值在两次调用函数之间将保持不变。

5. 每当函数被调用时，动态变量都被初始化；而静态变量只在函数首次调用时被初始化。

6. 不对。声明寄存器变量时，是在向编译器请求，编译器并不一定会满足这种请求。

7. 未被初始化的全局变量将自动被初始化为 0，但最好还是显式地进行初始化。

8. 未被初始化的局部变量将不会自动被初始化，其值是不确定的。决不要使用未被初始化的局部变量。

9. 由于变量 cout 的作用域为其所在的代码块，因此 printf()无法访问该变量。编译器将报错。

10. 如果值需要保留，则声明为静态的。例如，如果变量名为 vari，则声明如下：

```
static int vari;
```

11. 关键字 extern 被用作存储类型限定符，它指出变量已经在程序的其他地方声明过。

12. 关键字 static 用作存储类型限定符。它告诉编译器，保留变量的值。在函数中，这种变量的值在两次调用函数之间保持不变。

练习

1. register int x = 0;

2. 代码如下：

```
/* Illustrates variable scope. */
#include <stdio.h>

void print_value(int x);

int main( void )
{
        int x = 999;

        printf("%d", x);
        print_value( x );
        return 0;
}

void print_value( int x)
{
```

www.plcworld.cn


```
        printf("%d", x);
}
```

3. 由于 var 被声明为全局的，因此无需传递它。

```
/* Using a global variable */
#include <stdio.h>

int var = 99;

void print_value(void);

int main( void )
{
        print_value();
        return 0;
}

void print_value(void)
{
        printf( "The value is %d\n", var );
}
```

4. 是的。要在另一个函数中打印 var，必须将其传递给该函数。

```
/* Using a local variable*/
#include <stdio.h>

void print_value(int var);

int main( void )
{
        int var = 99;
        print_value(  var );

        return 0;
}

void print_value(int var)
{
        printf( "The value is %d\n", var );
}
```

5. 是的，程序中可以有同名的局部变量和全局变量。在这种情况下，处于可见状态的局部变量将覆盖全局变量。

```
/* Using a global */
#include <stdio.h>
int var = 99;

void print_func(void);

int main( void )
{
        int var = 77;
        printf( "Printing in function with local and global:");
        printf( "\nThe Value of var is %d", var );
```

www.plcworld.cn


```
        print_func( );

        return 0;
}
void print_func( void )
{
        printf( "\nPrinting in function  only global:");
        printf( "\nThe value of var is %d\n", var );
}
```

6. 函数 a_sample_function()只有一个错误。可以在任何代码块的开头声明变量，因此 ctrl 和 start 的声明没有问题。变量 ctr2 不是在代码块开头声明的，这是不对的。下面是更正后的函数，该函数被放到一个程序中。



注意：如果您使用的是C++编译器，而不是C语言编译器，则这个有问题的程序将通过编译，并能运行。对于变量的声明位置，C++中的规则不同于C语言。然而，您仍应遵循C语言的规则，即使您的编译器允许您不这样做。

```
#include <stdio.h>

void a_sample_function( );
int main( void )
{
        a_sample_function();
        return 0;
}

void a_sample_function( void )
{
        int ctrl;

        for ( ctrl = 0; ctrl < 25; ctrl++ )
                printf( "*" );

        puts( "\nThis is a sample function" );
        {
                char star = '*';
                int ctr2;     /* fix */
                puts( "\nIt has a problem\n" );
                for ( ctr2 = 0; ctr2 < 25; ctr2++ )
                {
                        printf( "%c", star);
                }
        }
}
```

7. 该程序能够正确运行，但有改进的余地。首先，无需将变量 x 初始化为 1，因为在 for 循环中，它被初始化为 0。另外，将变量 tally 声明为静态的毫无意义，因为在 main()函数中，关键字 static 无效。

8. star 的值是多少？dash 的值是多少？这两个变量没有被初始化。由于它们都是局部变量，因此包含的值是不确定的。注意，虽然该程序在编译时，没有出现错误或警告，但它仍然存在问题。

对于这个程序，还有另一个问题。变量 ctr 被声明为全局的，但只在函数 print_function()中使用了它。这种做法不好，应在函数 print_function()声明 ctr，这样它将是局部变量。

9. 这个程序不断地打印下述内容（参见练习 10）：

www.plcworld.cn


```
X==X==X==X==X==X==X==X==X==X==X==X==X==X==X==X==X==X==X==X==X==X==X==...
```

10. 该程序有一个问题，这是由于 ctr 是全局的。main()和 print_letter2()都在循环中将 ctr 作为计数器。由于 print_letter2()修改了 ctr 的值，因此 main()函数中的 for 循环永远不会结束。解决这种问题的方式很多。一种办法是使用不同的计数器变量；另一种方式是修改计数器变量 ctr 的作用域，可以分别在 main()和 print_letter2()中声明局部变量 ctr。

另外要说的是 letter1 和 letter2。这两个变量都只被用于一个函数中，因此应将它们声明为局部的。改正后的程序如下：

```
#include <stdio.h>
void print_letter2(void);          /* function prototype */

int main( void )
{
        char letter1 = 'X';
        int ctr;

        for( ctr = 0; ctr < 10; ctr++ )
        {
                printf( "%c", letter1 );
                print_letter2();
        }
        return 0;
}

void print_letter2(void)
{
        char letter2 = '=';
        int ctr;          /* this is a local variable */
                          /* it is different from ctr in main() */
        for( ctr = 0; ctr < 2; ctr++ )
                printf( "%c", letter2 );
}
```

第 13 天课程的答案

小测验

1. 永远不要使用（除非您非常小心）。

2. 遇到 break 语句后，立刻退出包含该语句的 for、do...while 或 while 循环。遇到 continue 语句后，立刻进入到下一次迭代。

3. 死循环将不断执行下去。创建的方法是，编写一个条件总为真的 for、do...while 或 while 循环。

4. 达到 main()函数末尾或调用 exit()函数后，程序将立刻结束。

5. switch 语句中的表达式的结果可以是 long、int 或 char 值。

6. default 语句是 switch 语句中的一个 case。当 switch 语句中的表达式的值不与任何 case 匹配时，将执行 default case。

7. exit()函数终止程序。可以将一个值传递给 exit()函数，这个值将被返回给操作系统。

8. system()函数执行一个操作系统命令。

www.plcworld.cn

练习

1. continue;
2. break;
3. 对于 DOS 操作系统，答案为：

```
system("dir");
```

4. 该代码片段没有错误。对于 case 'N'，无需在 printf()语句后加上 break 语句，因为此时总是会结束 switch 语句。
5. 您可能认为，default 应放在 switch 语句的最后，情况并非如此。default 可以放在任何地方。但该代码段还是有一个问题，default 的最后应有一条 break 语句。
6. 代码如下：

```
if( choice == 1 )
        printf("You answered 1");
else if( choice == 2 )
        printf( "You answered 2");
        else
                printf( "You did not choose 1 or 2");
```

7. 代码如下：

```
do {
        /* any C statements */
} while ( 1 );
```

第 14 天课程的答案

小测验

1. 流是一个字节序列。C 语言程序使用流来完成所有的输入/输出。
2.
 a. 打印机是输出设备；
 b. 键盘是输入设备；
 c. 调制解调器是输入和输出设备；
 d. 显示器是输出设备（虽然触摸屏既是输出设备，又是输入设备）；硬盘既可以是输入设备，也可以是输出设备。
3. 所有的编译器都支持三个预定义的流：stdin（键盘）、stdout（屏幕）和 stderr（屏幕）。有些编译器还支持 stdprn（打印机）、stdaux（串行口 COM1）。注意，Macintosh 不支持 stdprn。
4.
 a. stdout
 b. stdout
 c. stdin
 d. stdin
 e. fprintf()可以使用任何输出流。在 5 个标准流中，它可以使用 stdout、stderr、stdprn 和 stdaux。
5. 缓冲时，仅当用户按下 Enter 后，输入才被发送给程序；不缓冲时，用户按键后，相应的字符就被发送给程序。
6. 回显时，输入将被发送到 stdout；不回显时，则不会。
7. 只可以恢复一个字符，使用 unget()无法将 EOF 字符放回到输入流中。
8. 检查换行符，它表明用户按下了 Enter。
9.
 a. 合法：

www.plcworld.cn

b. 合法；

c. 合法；

d. 不合法，不存在说明符 q；

e. 合法；

f. 合法。

10. stderr 不能被重定向，它总是对应于屏幕；stdout 可以被重定向到除屏幕之外的其他设备。

练习

1. printf("Hello World") ;

2. fprintf(stdout, "Hello World");

```
puts( "Hello World");
```

3. fprintf(stdaux, "Hello Auxiliary Port");

4. 代码如下：

```
char buffer[31];
scanf( "%30[^*]", buffer );
```

5. 代码如下：

```
printf( "Jack asked, \"What is a backslash\?\"\nJill said, \
\"It is \'\\\'\"");
```

6. 提示：使用一个包含 26 个元素的 int 数组。要计算字符的数目，读取字符时，将相应的数组元素加 1。

7. 提示：每次读取一个字符串，然后打印格式化行号、制表符和这个字符串。提示 2：参见 Type & Run1 中的 Print_It 程序。

第 15 天课程的答案

小测验

1. 代码如下：

```
float x;
float *px = &x;
float **ppx = &px;
```

2. 错误是：该语句只使用一个间接运算符，这将把 100 赋给 px，而不是 x。该语句应使用两个间接运算符：

```
**ppx = 100;
```

3. array 是一个包含 2 个元素的数组，其中每个元素本身又是一个包含 3 个元素的数组，而这 3 个元素又都是包含 4 个元素的 int 数组。

4. array[0][0]是一个指针，指向第一个包含 4 个 int 元素的数组。

5. 第 1 和第 3 个表达式为 true，第 2 个表达式为 false。

6. void func1(char *p[]);

7. 无法知道，必须将这个值作为参数传递给函数。

8. 函数指针是一个变量，它存储的是函数在内存中的位置。

9. char (*ptr)(char *x[]);

10. 如果不使用圆括号将*ptr 括起，则这行代码将是一个函数的原型，该函数返回一个 char 指针。

11. 结构必须包含一个指针，该指针指向这种类型的结构。

12. 这意味着链表为空。

13. 链表的每个节点都包含一个指针，该指针指向链表中的下一个节点。链表的第一个节点由头指针标识。

14.

www.plcworld.cn

a. var1 是一个 int 指针；

b. var2 是一个 int 变量；

c. var3 是一个指向 int 指针的指针。

15.

a. a 是一个包含 36 个 int 元素的数组；

b. b 是一个指针，指向一个包含 12 个元素的 int 数组；

c. c 是一个包含 12 个元素的数组，其中的元素为 int 指针。

16.

a. z 是一个包含 10 个元素的数组，其中的元素为字符指针；

b. y 是一个函数，它接受一个 int 参数（field），并返回一个字符指针；

c. x 是一个函数指针，它指向的函数接受一个 int 参数（field）并返回一个字符指针。

练习

1. float (*func)(int field);

2. int (*menu_option[10])(char *title);

可结合使用函数指针数组和菜单系统。可以将菜单选项与函数指针数组的索引对应起来。例如，如果用户选择菜单中的第 5 个选项，则执行第 5 个数组元素指向的函数。

3. char *ptrs[10];

4. ptr 被声明为一个包含 12 个 int 指针的数组，而不是一个指针（指向一个包含 12 个元素的 int 数组）。正确的代码如下：

```
int x[3][12];
int (*ptr)[12];

ptr = x;
```

5. 下面是解决方案之一：

```
struct friend {
      char name[35+1];
      char street1[30+1];
      char street2[30+1];
      char city[15+1];
      char state[2+1];
      char zipcode[9+1];
      struct friend *next;
}
```

第 16 天课程的答案

小测验

1. 文本-模式流自动在换行符（\n，C 语言使用它来标记行尾）和回车换行符（DOS 使用它来标记行尾）之间进行转换；而二进制-模式流不执行这样的转换，所有的字节都按原样输入和输出。

2. 使用库函数 fopen()打开文件。

3. 使用 fopen()时，必须指定要打开的磁盘文件的名称以及打开模式。函数 fopen()返回一个 FILE 指针，接下来调用文件存取函数时，将使用该指针来引用文件。

4. 格式化、字符和直接。

www.plcworld.cn

5. 顺序和随机。

6. EOF 是文件尾标记，它是一个符号常量，值为-1。

7. 确定是否到达文本文件的末尾。

8. 在二进制模式下，必须使用函数 feof()；在文本模式下，可以使用 EOF 字符或 feof()函数。

9. 文件位置指示器指出在文件的什么位置进行读写。可以使用 rewind()和 fseek()来修改文件位置指示器。

10. 文件位置指示器指向文件开头（偏移量为 0）。一种例外情况是，如果以追加模式打开一个已有的文件，则文件位置指示器将指向文件尾。

练习

1. fcloseall();

2. rewind(fp);和 fseek(fp, 0, SEEK_SET);

3. 不能使用 EOF 来判断是否到达二进制文件的末尾，而应该使用函数 feof()。

第 17 天课程的答案

小测验

1. 字符串的长度指的是从字符串开头到结尾的空字符之间的字符数（不包括空字符），可以使用函数 strlen()来确定字符串的长度。

2. 必须分配足够的存储空间来存储新的字符串。

3. 拼接指的是将两个字符串合并起来——将一个字符串追加到另一个字符串的后面。

4. 对字符串进行比较时，"大于"指的是一个字符串的 ASCII 值大于另一个字符串的 ASCII 值。

5. strcmp()函数对两个字符串的全部字符进行比较；而 strncmp()只比较字符串中的前几个字符。

6. strcmp()比较字符串时，区分字母的大小写（例如，A 和 a 是不同的）；strcmpi()不考虑大小写（例如，A 和 a 是相同的）。

7. isascii()检查参数是否是 ASCII 值为 0～127 的标准 ASCII 字符，而不检查扩展 ASCII 字符。

8. isascci()和 iscntrl()返回 True，其他宏返回 False。别忘了，这些宏检查的是字符值。

9. 65 相当于 ASCII 字符 A。下述宏返回 True：isalnum()、isalpha()、isascii()、isgraph()、isprint()和 isupper()。

10. 字符检测函数确定特定的字符是否满足某种条件，如是字母、标点还是其他东西。

练习

1. TRUE（1）或 FALSE（0）。

2.

 a. 65

 b. 81

 c. –34

 d. 0

 e. 12

 f. 0

3.

 a. 65.000000

 b. 81.230000

 c. –34.200000

 d. 0.000000

www.plcworld.cn

e. 12.000000

f. 1000.000000

4. 使用 string2 之前，没有将它指向某个位置，无法确定 strcpy()将把 string1 的值复制到什么地方。

第 18 天课程的答案

小测验

1. 按值传递意味着函数将收到参数变量的值的一个拷贝；按引用传递意味着函数收到的将是参数变量的地址。它们之间的差别在于，按引用传递使函数能够修改原来的变量；而按值传递不能。

2. void 指针可以指向任何数据对象，换句话说，它是一个通用指针。

3. void 指针是通用指针，可以指向任何对象。void 指针常被用来声明函数参数。您可以创建能够处理不同类型参数的函数。

4. 强制类型转换指出 void 指针指向的数据对象的类型。解除 void 指针的引用之前，必须将其强制转换为特定的类型。

5. 接受可变数目参数的函数必须至少有一个固定参数。该参数用于指出调用函数时，传递了多少个参数。

6. 应该使用 va_start()来初始化参数列表，使用 va_arg()来取回参数，取回所有的参数后，应使用 va_end()来完成清理工作。

7. 这是一个恶作剧式问题！不能对 void 指针执行递增运算，因为编译器不知道应将它的值增加多少。

8. 函数可以返回指向任何数据类型的指针，还可以返回指向数组、结构和共用体的指针。

9. va_arg()。

10. 元素 va_list、va_start()、va_arg()和 va_end()。

练习

1. int function(char array[]);

2. int numbers(int *nbr1, int *nbr2, int *nbr3);

3. 代码如下：

```
int number1 = 1, number2 = 2, number3 = 3;
numbers( &number1, &number2, &number3);
```

4. 虽然这些代码令人迷惑，却是正确的。这个函数接受 nbr 指向的值，并计算其平方。

5. 使用可变参数列表时，应使用所有的宏工具。这包括 va_list()、va_start()、va_arg()和 va_end()。有关如何正确地使用可变参数列表，请参阅程序清单 18.3。

第 19 天课程的答案

小测验

1. double。

2. 在大多数编译器中，它与 long 等价，但并非总是如此。要了解您的编译器使用的是什么类型，可查看文件 TIME.H 或编译器用户参考手册。

3. 函数 time()返回从 1970 年 1 月 1 日午夜起过去的秒数；函数 clock()返回从程序开始执行起过去的时间（单位为 1/100 秒）。

4. 不会，只是显示一条描述错误的消息。

www.plcworld.cn

5. 将数组元素按升序排列。

6. 14。

7. 4。

8. 21。

9. 如果相等，则返回 0；如果第一个值大于第二个值，则返回一个大于 0 的值；如果第一个值小于第二个值，则返回一个小于 0 的值。

10. NULL。

练习

1. 代码如下：

```
bsearch( myname, names, (sizeof(names)/sizeof(names[0])),
sizeof(names[0]), comp_names);
```

2. 存在三个问题。首先，调用 qsort()时没有提供字段宽度（field width）；其次，调用 qsort()时，函数名后不应包含圆括号；最后，程序没有提供比较函数。qsort()使用 compare_function()来进行比较，但程序中没有定义该函数。

3. 比较函数的返回值不正确。当 element1 大于 element2 时，它应返回一个正值；而当 element1 小于 element2 时，它应返回一个负值。

第 20 天课程的答案

小测验

1. malloc()分配指定数目字节的内存，而 calloc()为指定数目的数据对象分配足够的内存。另外，calloc()还将内存中的字节全部设置为 0，而 malloc()不会执行任何初始化工作。

2. 将两个整型值相除，并将结果赋给浮点型变量时，保留结果中的小数部分。

3.

a. long

b. int

c. char

d. float

e. float

4. 动态内存分配是在程序运行时进行的，它使您能够根据需要分配内存。

5. 当源和目标内存区域相互重叠时，memmove()能够正确地工作，而 memcpy()不行。当源和目标内存区域不相互重叠时，这两个函数的功能相同。

6. 定义一个长度为 3 位的位字段成员。由于 2^3 为 8，因此该字段能够存储 1～7 的值。

7. 2 个字节，使用位字段，可以声明一个这样的结构：

```
struct date
{
    unsigned month : 4;
    unsigned day   : 5;
    unsigned year  : 7;
}
```

这个结构使用 2 个字节（16 位）来存储日期。4 位的 month 字段可以存储 0～15 的值，因此足以表示 12 个月份。同样，5 位的 day 字段可以存储 0～31 之间的值，而 7 位的 year 字段可以存储 0～127 之间的值。假设将加上 1900 来获得年份，因此可表示的年份为 1900～2027。

8. 00100000。

www.plcworld.cn

9. 00001001。

10. 这两个表达式的结果相同。同 11111111 进行异或运算与求反的效果相同：每一位都被反转。

练习

1. 代码如下：

```
long *ptr;
ptr = malloc( 1000 * sizeof(long));
```

2. 代码如下：

```
long *ptr;
ptr = calloc( 1000, sizeof(long));
```

3. 使用循环和赋值语句：

```
int count;
for (count = 0; count < 1000; count++)
data[count] = 0;
```

使用 memset()函数：

```
memset(data, 0, 1000 * sizeof(float));
```

4. 这些代码能够编译并运行，但结果不正确。由于 number1 和 number2 都是 int 变量，因此它们的结果也为 int 类型，小数部分将丢失。要获得正确答案，必须将表达式强制转换为 float 类型：

```
answer = (float) number1/number2;
```

5. 由于 p 是一个 void 指针，因此将其用于赋值语句中之前，必须强制转换其类型。第 3 行应为：

```
*(float*)p = 1.23;
```

6. 不允许。位字段必须位于结构的最前面。正确的代码如下：

```
struct quiz_answers
{
    unsigned answer1   : 1;
    unsigned answer2   : 1;
    unsigned answer3   : 1;
    unsigned answer4   : 1;
    unsigned answer5   : 1;
    char student_name[15];
}
```

第 21 天课程的答案

小测验

1. 模块化编程指的是一种将程序分为多个源代码文件的程序开发方法。
2. 主模块包含 main()函数。
3. 通过确保传递给宏的复杂表达式被首先计算，避免不希望的副作用。
4. 与函数相比，宏使得程序的执行速度更快，但程序的大小更大。
5. 运算符 defined()检查某个名称是否已被定义。如果已被定义，则返回 TRUE，否则返回 FALSE。
6. 必须使用#endif。
7. 编译后的源代码是扩展名为.obj 的目标文件。
8. #include 将另一个文件的内容复制到当前文件中。
9. 使用双引号时，将在当前目录中查找包含文件；使用< >时，将在标准目录中查找包含文件。
10. __DATE__用于将编译日期加入到程序中。

www.plcworld.cn

11. 一个包含当前程序的名称（包括路径信息）的字符串。

附加课程 1 的答案

小测验

1. OOP 语言的特征如下：

- 多态；
- 封装；
- 继承；
- 重用。

2. 多态。

3. 所有的 C 语言关键字都可用于 C++程序中。

4. 可以使用 C 语言进行面向对象编程，但非常复杂。C++和 Java 都具备面向对象特性。要进行面向对象编程，使用 C++或 Java 将好得多。

5. 否。Java 删除了 C++中那些导致不必要的复杂性和容易引起错误的特性。

6. 对于 C++，答案是肯定的——您可以在程序中使用过程性技术或面向对象技术；对于 Java，答案是否定的——必须采用面向对象技术。

7. cout。

8. System.out.println()。

9. System.Console.WriteLine()。

附加课程 2 的答案

小测验

1. cout。

2. 绝不要使用。

3. bool。

4. 在 C++中，可以在程序的任何位置声明变量。

5. throw。

6. 默认值让函数可以接受不同数目的参数；重载让您能够创建多个同名的函数，它们接受的参数数目和类型不同。

7. 这三个函数的参数数目和类型都相同，因此不能彼此重载。

8. int triangle(int side1 = 0, int side2 = 0, int side 3 = 0);

9. 这个问题有些恶作剧。这里对内联函数的定义是正确的，但并非对于每个被声明为内联的函数，编译器都会这样做。如果编译器认为，这样做的效率不会更高，则可能忽略说明符 inline。

附加课程 3 的答案

小测验

1. 在 C++中，结构是一种特殊的类。默认情况下，结构的成员是公有的，而类的成员是私有的。如果需要有

www.plcworld.cn

成员函数，应使用类，而不是结构。

2. 类只是一种定义，因此不能给它赋值。类用于创建对象，而您可以给对象赋值。

3. 对象是一个使用类创建的变量。

4. 实例化类指的是使用这个类来创建对象。

5. 类具备所有的三种面向对象特征。类将所有的数据和功能组合在一起，因此具备封装特征；类可以包含重载函数（包括构造函数），因此具备多态特性；类还具备继承特征，附加课程 4 将详细介绍继承。

6. 私有数据成员只能通过同一个类的成员函数来访问。

7. 公有数据成员可在程序的任何地方直接进行访问。

8. 对象被创建时将调用构造函数。

9. 对象被释放时将调用析构函数。

10. 用作数据成员时，类的用法和其他数据类型没有任何区别。可以像使用整型、字符、结构或其他任何数据类型一样使用类。

11. b。

12. a。

13. class guppy : public fish{

14. 首先执行基类的构造函数。

15. 首先执行子类的析构函数。

附加课程 4 的答案

小测验

1. long。

2. 支持，但它们被称作方法。

3. while 或 do...while。

4. 以便在 Java 程序中使用其他类。

5. Java 注释风格有三种：

- /*和*/之间的内容为注释；
- //到行尾的内容为注释；
- /**和*/之间的内容为注释。

6. Java 标识符是区分大小写的，因此 count 和 Count 是两码事。

7. 任何数据类型——对数组的数据类型没有任何限制。

附加课程 5 的答案

小测验

1. 您没有选择余地。构造函数的名称总是与类名相同。

2. 在类定义的第一行使用关键字 extends 来指定新类将继承的父类。

3. 不。如果方法的返回类型被声明为 void，则该方法不会返回任何值。

4. 是的，只要参数的类型和/或数目不同即可。

5. 创建类的实例时。

6. 不。如果定义类时使用了关键字 final，则它不能用作父类。

www.plcworld.cn

7. 关键字 final 用于防止类被继承。
8. 不，抽象类不能用来创建对象；但可以被继承。

附加课程 6 的答案

小测验

1. 是的。如果 try 语句块没有相应的 catch 语句块，Java 编译器将报错。
2. 不会。如果要先换行，必须在字符串前面加上换行字符\n。
3. java.applet.Applet。
4. show 方法。
5. 响应用户事件的技术有多种，但最重要的一种是使用 action 方法，当诸如鼠标单击等事件发生时，该方法将自动被调用。
6. 可以，但它作为小程序运行时将被忽略。
7. init 方法和 start 方法。
8. destroy 方法。

附加课程 7 的答案

小测验

1.

- C#是简单的；
- C#是现代的；
- C#是面向对象的；
- C#功能强大而灵活；
- C#使用的单词不多；
- C#是流行的。

2. IL 代表中间语言（Intermediate Language）；CLR 代表通用语言运行阶段环境（Common Language Runtime）。
3.
 a. 创建源代码；
 b. 编译程序；
 c. 执行程序。
4. csc my_prog.cs
5. .cs。
6. 是的，但不推荐这样做。
7. 应核对代码，确保没有逻辑错误。
8. 计算机能够理解的语言。
9. Write()和 WriteLine()。

练习

1. 程序清单可能晦涩而混乱，包含一些怪字符。
2. 该程序打印一个由 X 组成的长方形。

www.plcworld.cn

XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXX

www.plcworld.cn

附录 G　Dev-C++编译器

本书附带的光盘中有一个 Dev-C++的拷贝。虽然这是一个 C++编译器，但也可用来创建 C 语言程序。本附录介绍以下内容：

- Dev-C++简介；
- 如何安装 Dev-C++？
- 在 Dev-C++中，如何输入、编译和运行程序？



> 警告：Bloodshed Dev-C++是一个 Windows 程序。如果您的操作系统不是 Windows 95 或更高的版本，将无法使用该程序。更详细的信息，请参阅该编译器自带的文档。

G.1　Dev-C++简介

Bloodshed Dev-C++是一个集成开发环境（IDE），可用于创建 C 或 C++程序。本书附带的光盘中包含这个编译器。

Dev-C++有很多工具，如安装程序创建器（Setup Creator）、自动生成 DLL、工程模板（Project Templates）、图标库（Icon Library）等。有关这些工具更详细的信息，请参阅这些工具自带的帮助文档。

要使用 Dev-C++，您的操作系统必须是 Windows 95 或更高的版本（98、2000、NT 等）。另外，您的系统必须至少有 8MB 的内存、30MB 的可用硬盘空间和 100MHz 的处理器。为获得更高的性能，建议系统有 32MB 的内存、45MB 的可用硬盘空间以及 233MHz 的处理器。

G.2　在 Microsoft Windows 上安装 Dev-C++

本书附带的光盘中包含了 Dev-C++。如果您的系统启用了自动运行功能，则当您插入该光盘后，将出现一个菜单。选择 Launch Dev-C++ Installer 将开始安装 Dev-C++。

如果没有启用自动运行功能，可直接运行该光盘中的安装程序，方法是选择并运行 Dev-C++目录中的 Setup.exe 程序。如果您的光驱为 D:盘，则命令为 D:\Dev-C++\Setup.EXE。

不管采用哪种方式来安装 Dev-C++，都将首先出现许可协议对话框（如图 G.1 所示）。

Dev-C++是通过标准的 GNU 许可协议分发的。如果您不熟悉 GNU 许可协议，请阅读图 G.1 的对话框中的信息。

从中可以选择安装类型。如果不知道应选择哪种类型，可选择典型（Typical）安装。在该对话框中，也可以修改 Dev-C++的安装位置。默认情况下，该程序被安装到 C 盘的 Dev-C++目录中。如果要将其安装到其他地方，可以单击 Browse 按钮，并输入新的目录。设置完毕后，可单击 Next 按钮，进入下一步。

www.plcworld.cn

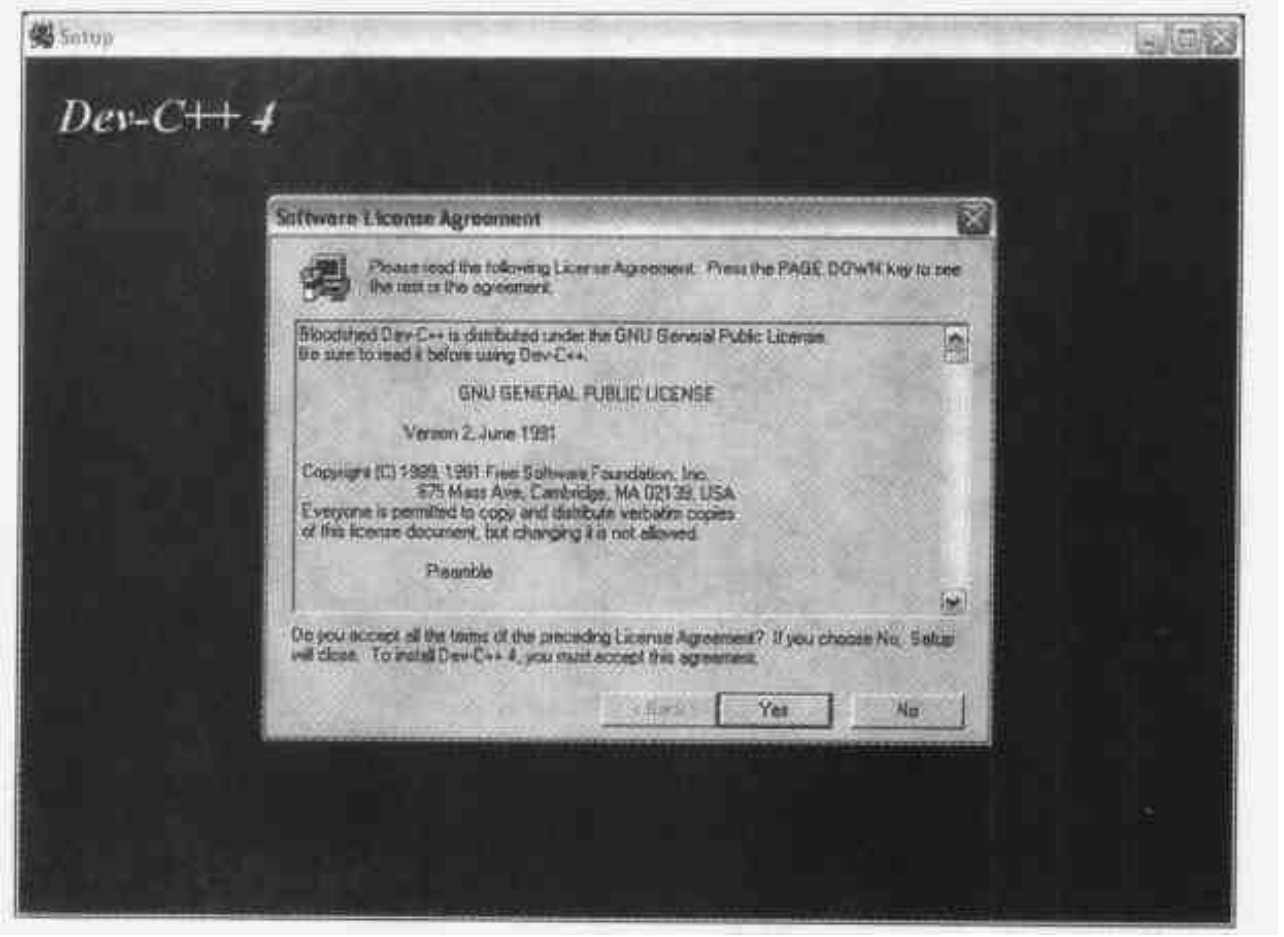


图 G.1　Dev-C++许可协议

单击 Yes 按钮，可以进入下一步——如图 G.2 所示的对话框。

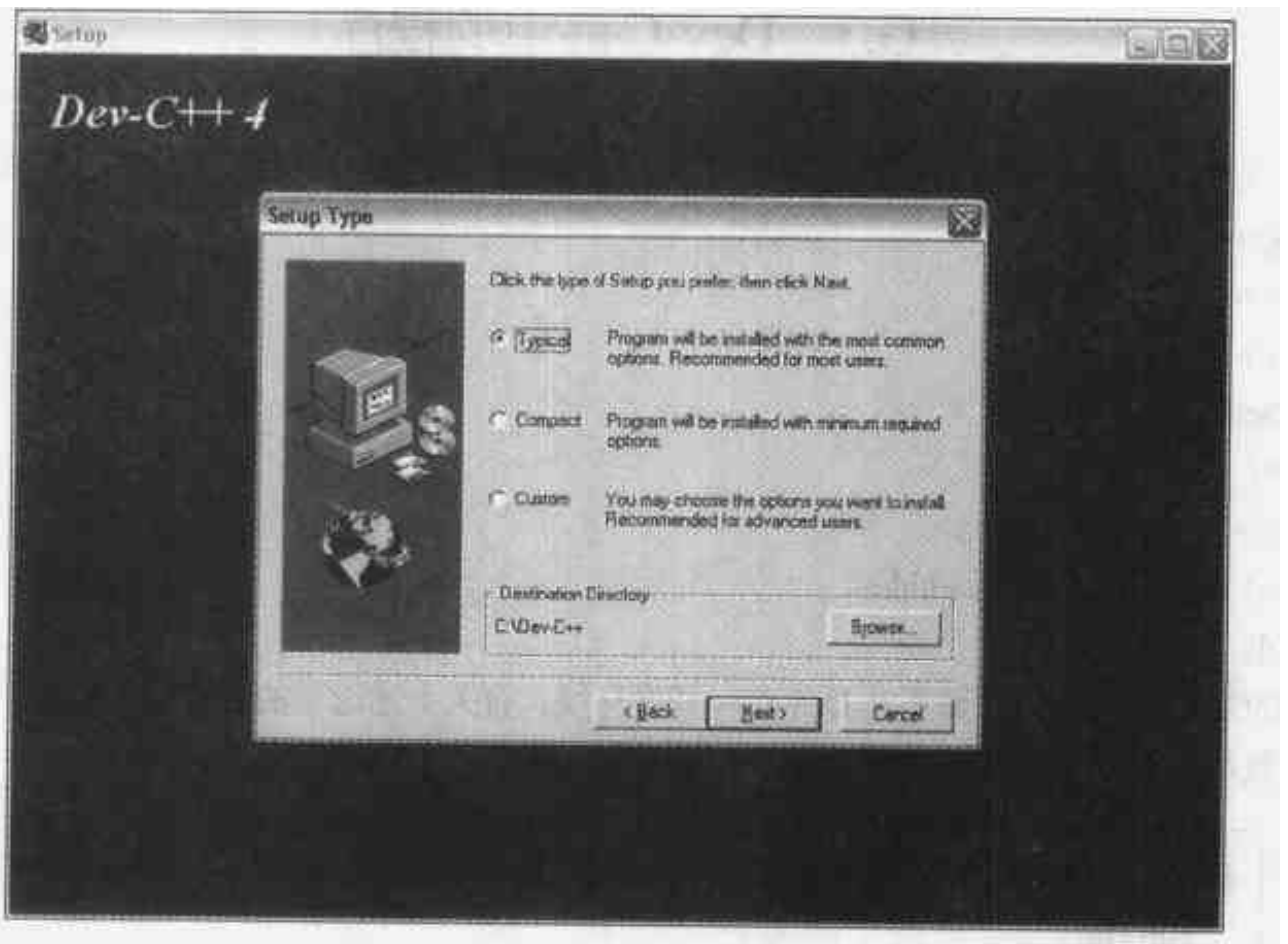


图 G.2　Setup Type 对话框



注意：您也可以选择安装选项 Compact 或 Custom。Compact 选项让您安装最少的组件，因此占用的空间最小。Custom 选项让您能够控制安装哪些组件。

选择安装类型后，将开始安装工作，如图 G.3 所示。

复制所有的文件后，安装程序将把 Dev-C++加入到“开始”菜单中，然后显示最后一个对话框（如图 G.4 所示）。通过该对话框，可以查看 readme 文件和启动该程序。另外，也可以单击 Finish 按钮，结束安装。然后，便可以通过 Windows 的“开始”菜单来运行该程序。

www.plcworld.cn

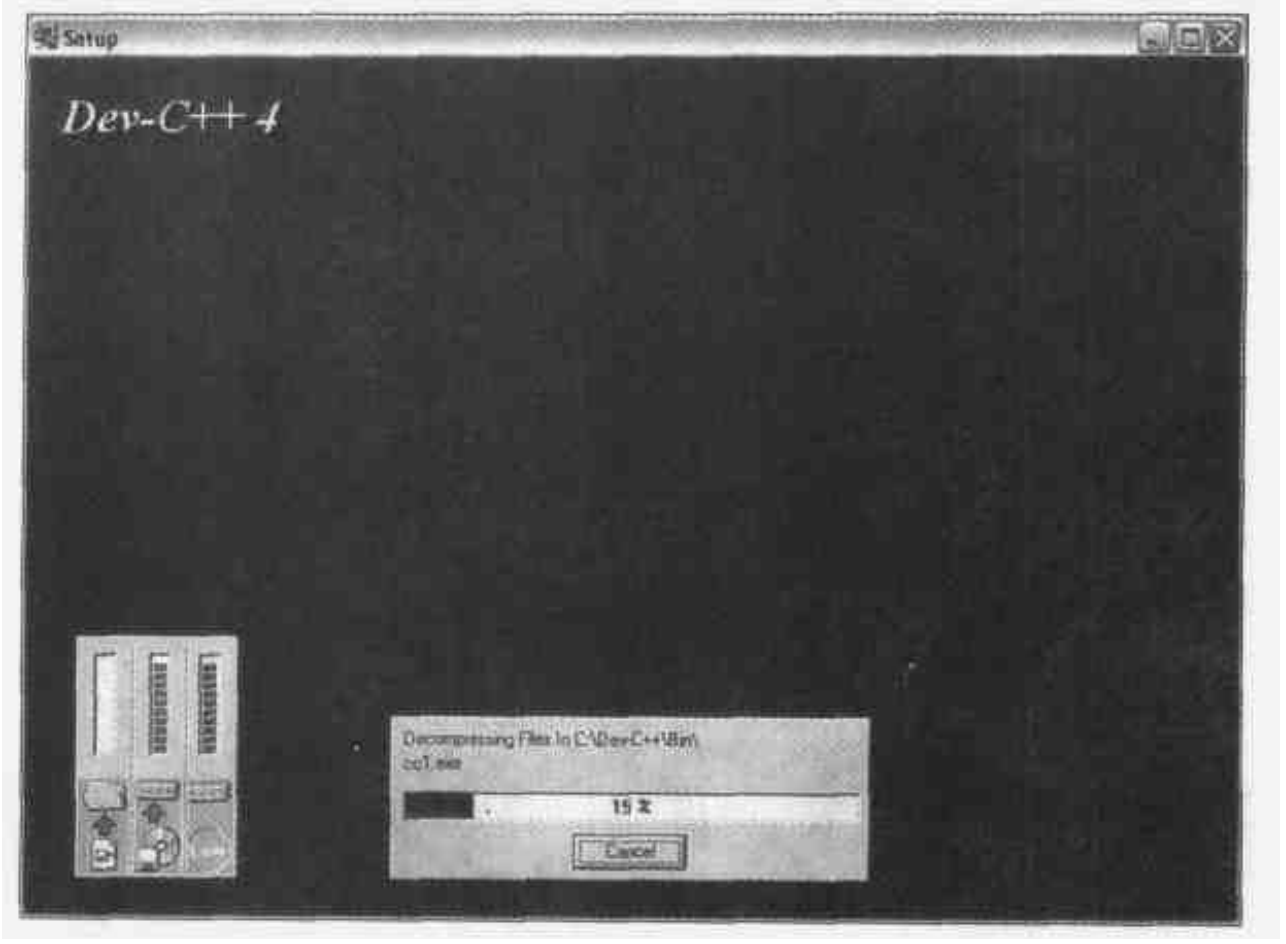


图 G.3　安装 Dev-C++文件

G.3　Dev-C++中的程序

Dev-C++安装程序在 Windows“开始”菜单中添加很多链接。Dev-C++菜单项中包含以下链接：

- Debugger；
- Dev-C++；
- Dev-C++ help file；
- GDB Debugger help；
- License；
- ReadMe；
- Standard Template Library guide；
- Tutorial。

本书介绍的是 C 语言编程，而不是 Dev-C++开发环境。有关上述选项的更详细的信息，请参阅 Dev-C++的帮助文件和教程。接下来的几节将简要地介绍如何使用 Dev-C++来编译 C 语言程序。



提示：如果您要更详细地了解 Dev-C++，请首先阅读 Dev-C++的帮助文档，而不是教程（Tutorial）。

G.4　使用 Dev-C++

Dev-C++可用来创建 C 语言程序和 C++程序。要运行这种开发环境，可选择菜单“开始/程序/Dev-C++/Dev-C++，这将启动 Dev-C++ IDE，进入如图 G.4 所示的 IDE。

虽然 Dev-C++通常使用工程文件，但您也可以使用它来输入单个 C 语言源代码文件。工程文件用于创建

www.plcworld.cn

可包含多个文件的应用程序。对于本书创建的大多数应用程序，都只需输入一个源代码文件。

G.4.1　针对 C 语言编程定制 Dev-C++

在介绍如何输入和运行程序之前，先介绍几个需要设置的选项。这些选项只需设置一次。

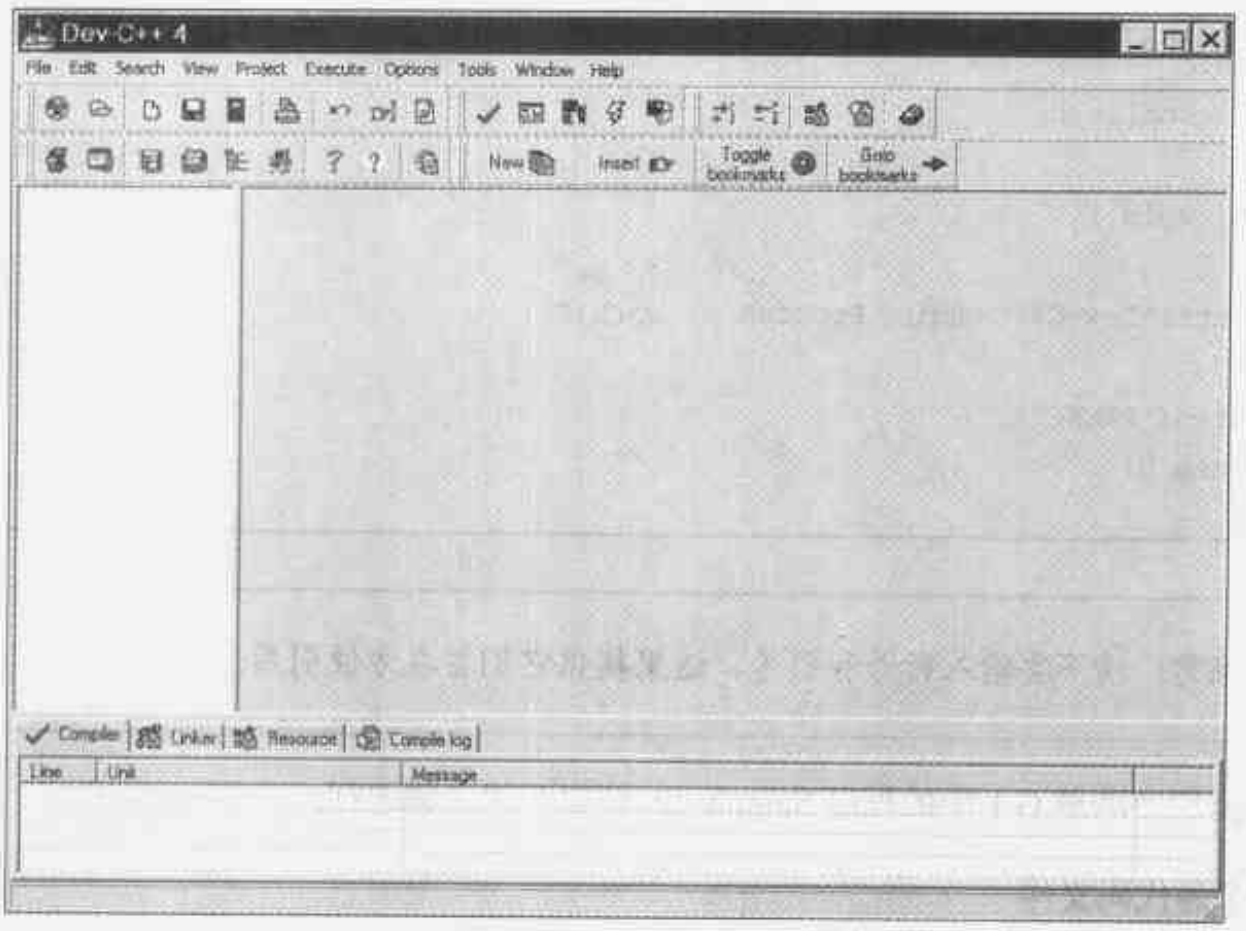


图 G.4　Dev-C++ IDE

要设置编译器，使之遵循 ANSI C 标准，请选择 Options 菜单中的 Compiler options 选项，打开如图 G.5 所示的对话框。

然后，单击标签 C/C++ Compiler，该对话框将变成如图 G.6 所示。

在图 G.6 的对话框中，首先选中复选框 Support all ANSI C programs，它使编译器能够识别 ANSI 标准命令；另外，还应选中复选框 Attempt to support some aspects of traditional C compilers。有关这些选项的更详细的信息，请参阅该编译器自带的帮助文档。

请注意，Dev-C++编译器支持的是 ANSI C-99（ISO/IEC 9899:1999）之前的标准，这意味它不能识别 ANSI 标准中新增的内容，其中包括对单行注释的支持。选中了复选框 Support all ANSI C programs 后，如果使用单行注释，编译器将报错。因此，如果您要使用单行注释，则不应选中该选项。有关不支持的特性，请参阅 Dev-C++ 中的帮助文档。

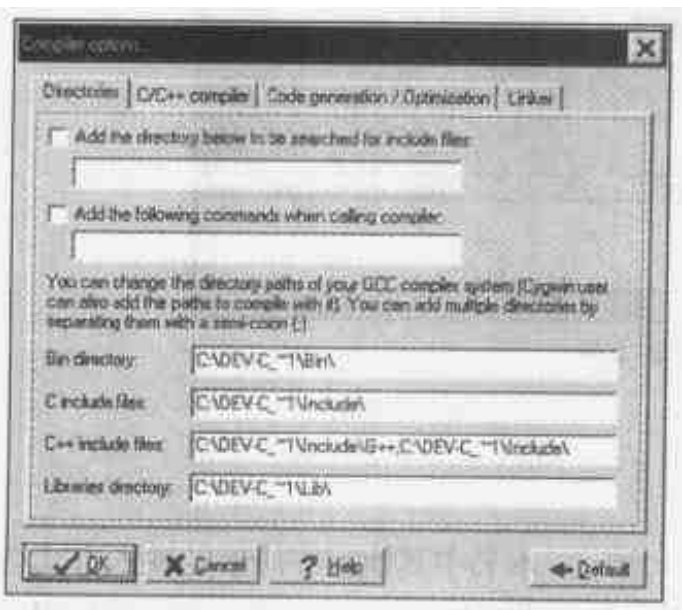


图 G.5　Compiler options 对话框

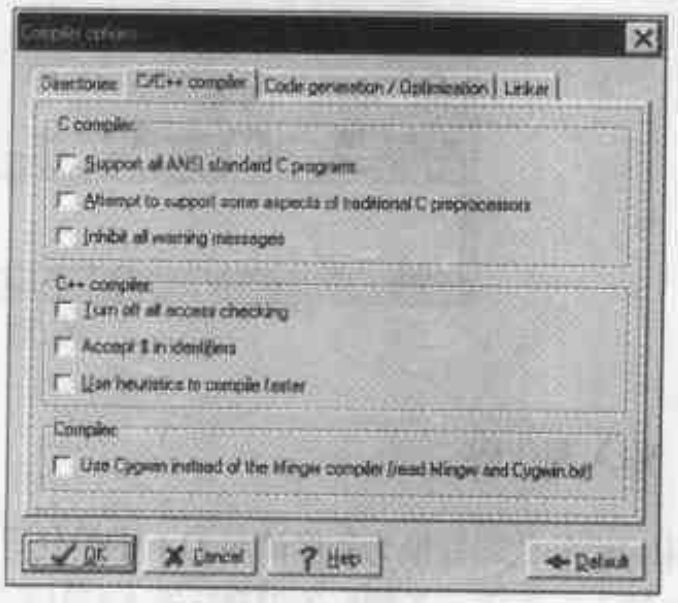


图 G.6　C/C++ options 对话框

www.plcworld.cn

G.4.2 在 Dev-C++中输入并编译程序

对于本书中的程序，您无需在 Dev-C++中使用工程，而可以分别输入并编译每个程序。程序清单 G.1 是一个小型程序，您可以输入并编译它。

程序清单 G.1　　Sample.c：范例程序

```
1:  // Sample.c
2:  #include <stdlib.h>
3:
4:  int main( void )
5:  {
6:      printf("Dev-C++ Sample Program at work!");
7:
8:      system("PAUSE");
9:      return 0;
10: }
```



注意：请不要输入行号和引号，这里提供它们旨在方便引用。

要输入并编译程序清单 G.1 中的程序 Sample.c，请按下述步骤进行：

1．新建一个源代码文件

打开 Dev-C++后，选择菜单 File/New Source file，Dev-C++将自动创建一个源代码文件，并在其中放置一些代码。图 G.7 是包含默认代码的窗口。

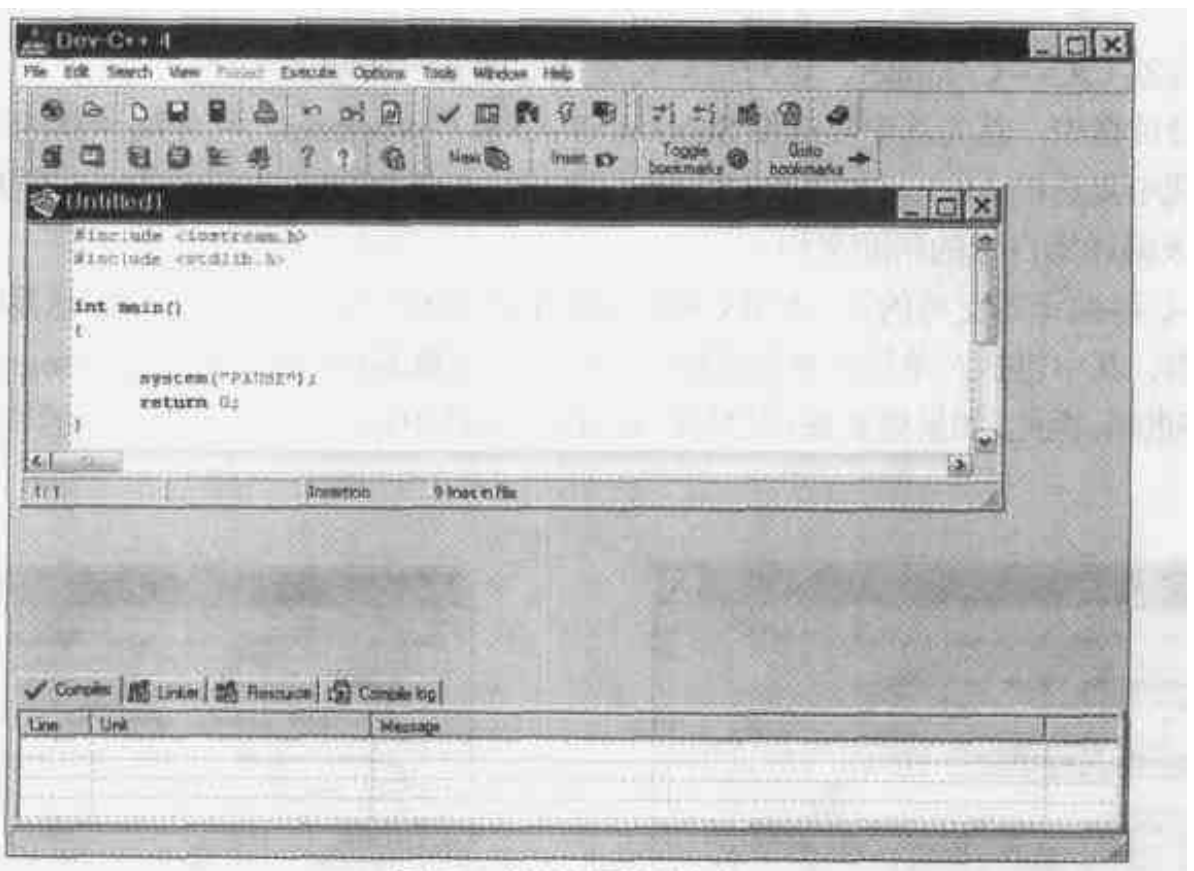


图 G.7　默认的源代码文件

2．输入源代码

将窗口中的代码替换为程序清单 G.1 中的代码，如图 G.8 所示。本书中其他程序清单的输入方式与此相同。该程序的第 6 行打印一条简单的消息，第 8 行让程序暂停，直到用户按下 Enter。如果没有这一行代码，则执行程序时，用户看不到结果，因为结果将被显示，然后迅速地消失。有关这一点，后面将做更详细的说明。

www.plcworld.cn

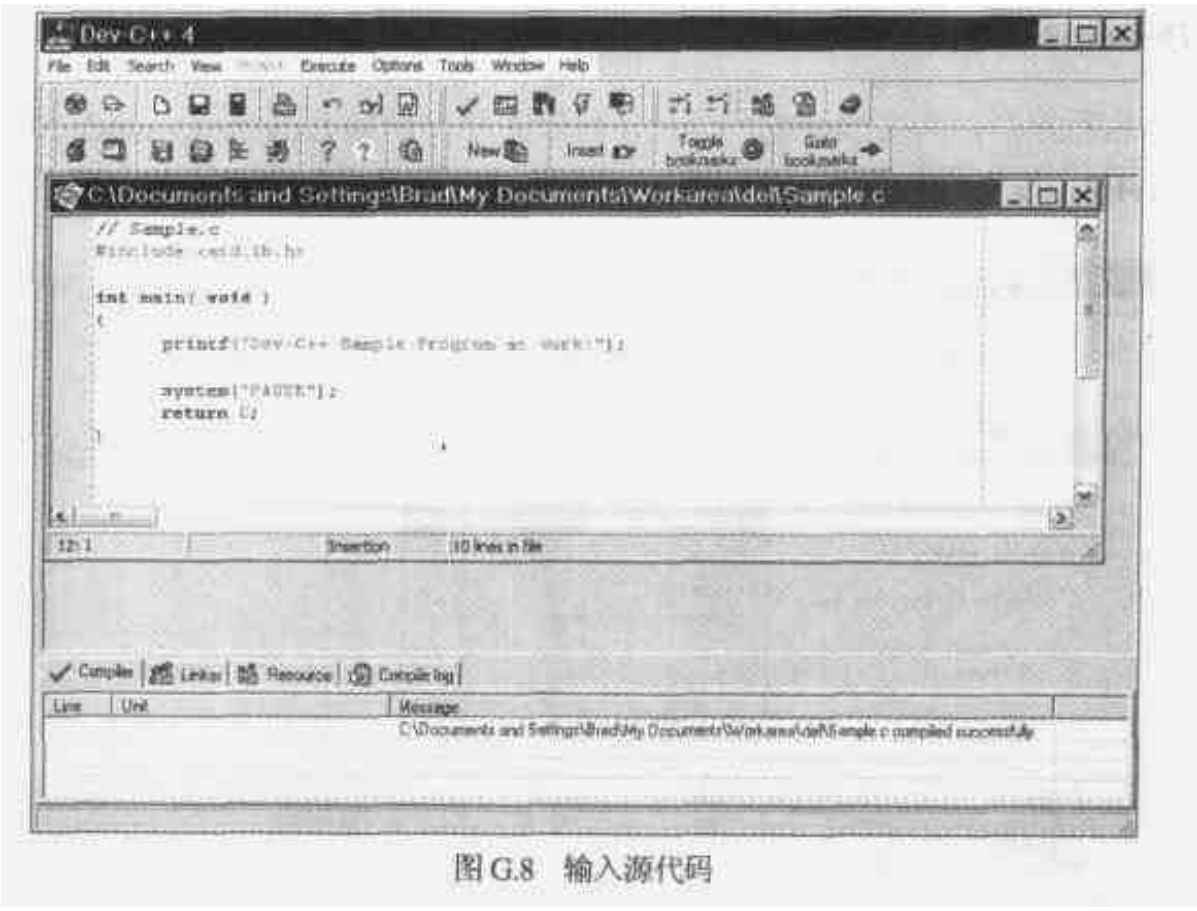


图 G.8　输入源代码

3. 保存源代码文件

输入源代码后，要保存它。Dev-C++提供的默认文件名由 Untitled 和一个数字组成。要保存程序，并给它提供一个更有意义的名称，可选择 File/Save unit as 菜单。开发环境将显示一个标准的文件保存对话框，如图 G.9 所示。

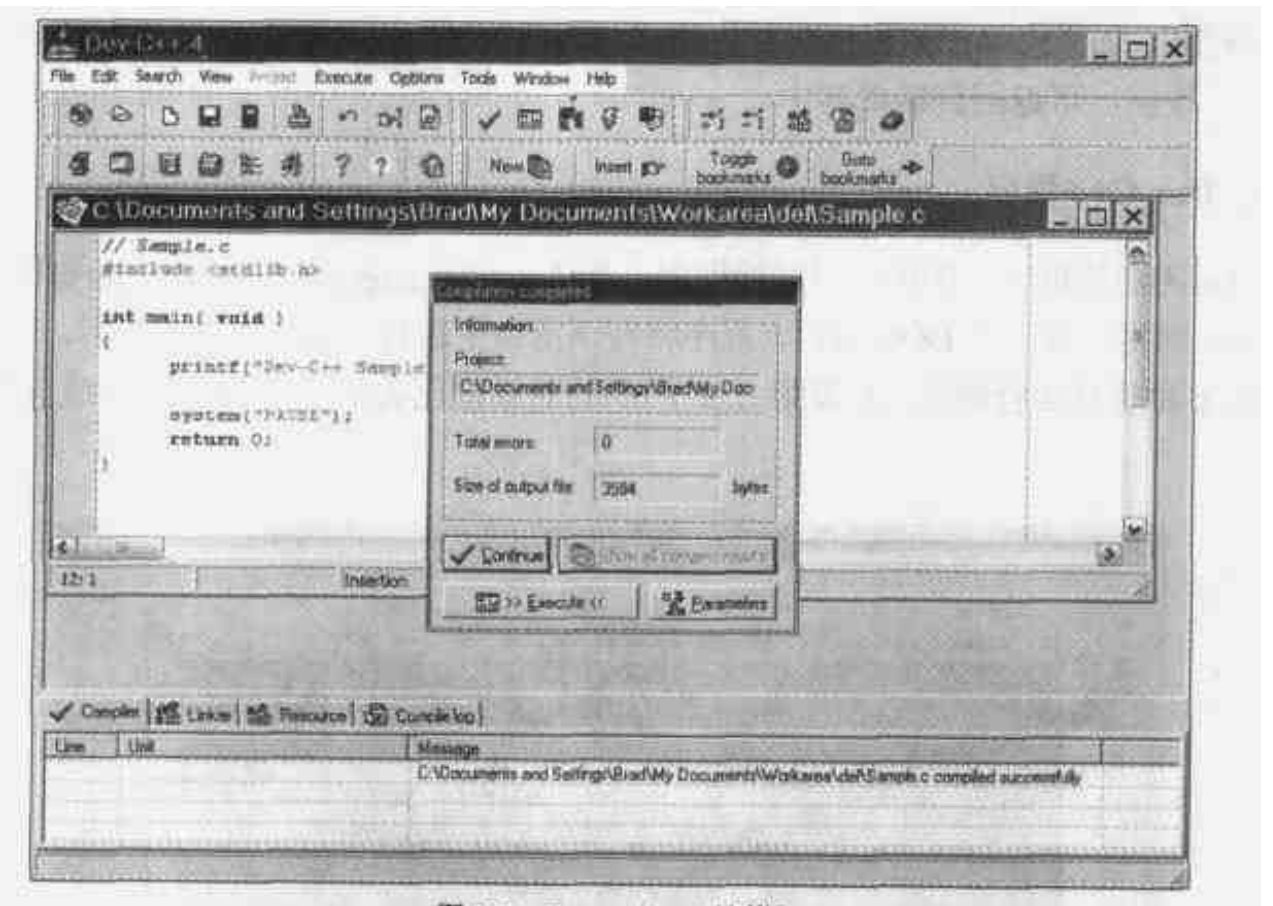


图 G.9　Save unit as 对话框

在该对话框中，您可以指定文件的名称，也可以修改文件的存储目录。保存文件之前，应将 Save source as 选项设置为C source file，这将确保扩展名为.c而不是.cpp。请将范例文件命名为 Sample。通过选择选项C source file，开发环境会自动加上扩展名.c。保存文件后，对话框中将显示新的文件名。以后，您可以通过单击工具栏中的磁盘图标按钮来保存所做的修改。



注意：实际上，磁盘图标按钮和 File/Save unit as 菜单项的功能是相同的。

www.plcworld.cn

G.4.3 编译 Dev-C++程序

输入并保存 C 语言程序后，需要编译它。为此，可以选择菜单项 Execute/Compile、单击工具栏上的复选标记按钮或按下 Ctrl+F9。编译完程序后，将出现一个对话框，指出是否有错误，如图 G.10 所示。

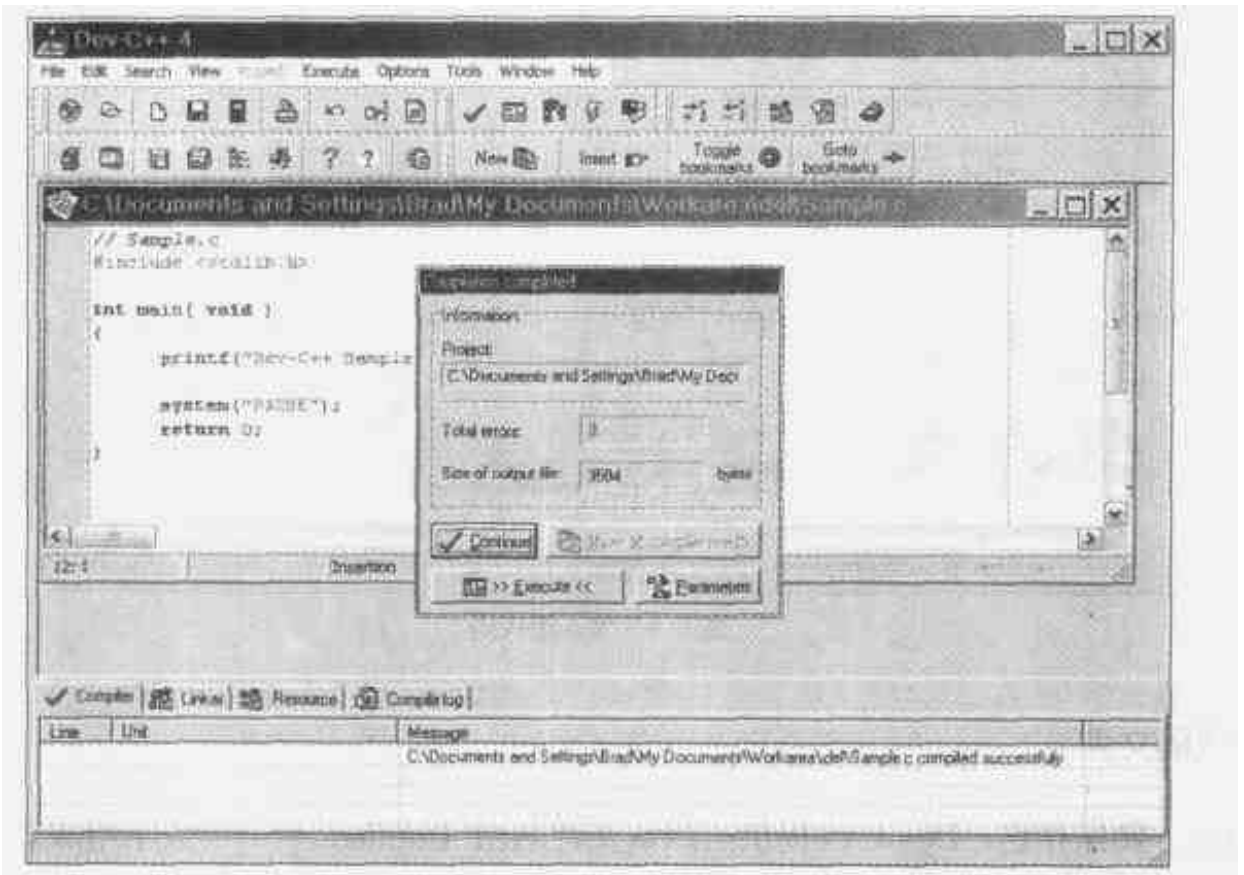


图 G.10 编译程序

从图 G.10 中的 Total errors 文本框可知，没有错误。如果该文本框的值不为 0，则说明有错误，而这些错误将在屏幕底部列出。如果您编译该程序时发生错误，请检查输入是否正确，且没有输入行号和冒号。

单击 Continue 按钮，可返回到编辑器中。

G.4.4 运行 Dev-C++程序

在图 G.10 所示的对话框中，您也可以运行程序，方法是单击 Execute 按钮。如果程序像程序清单 G.1 那样，是 DOS 程序，则将打开一个 DOS 窗口，程序将在该窗口中运行。程序运行完毕后，该窗口将自动关闭。图 G.11 是 Sample.c 程序的运行情况。如果图 G.10 所示的对话框已关闭，则可以按 F9 或选择菜单 Excute/Run 来运行程序。

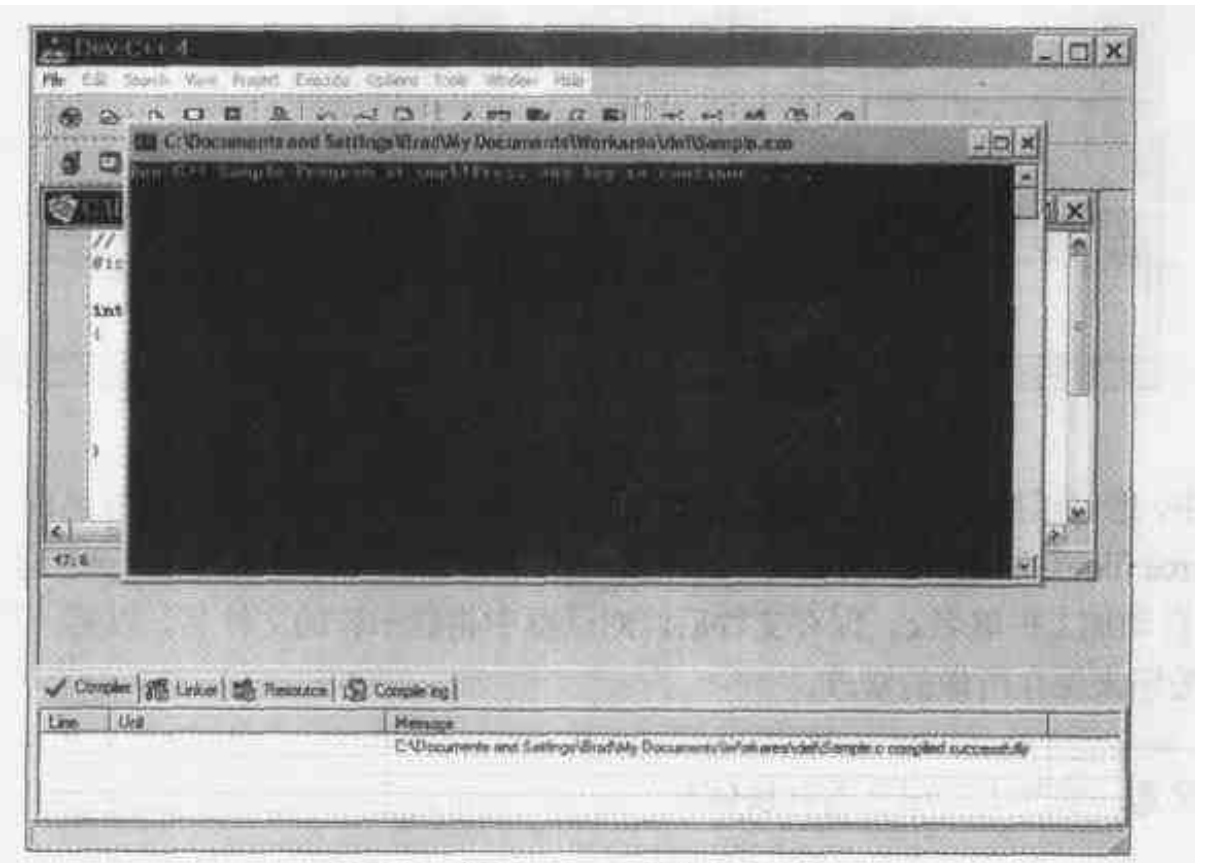


图 G.11 在 Dev-C++ IDE 中运行程序

www.plcworld.cn

第 8 行的 system("PAUSE")命令使程序暂停，直到用户按下某个键。本书中的大多数程序都不包含这个命令。这意味着您可能看不到程序的运行情况，这是因为系统将首先打开一个 DOS 窗口，运行程序，然后立刻关闭该窗口。为避免这种情况，您可以打开一个 DOS 窗口，方法是选择菜单“开始/程序/MS-DOS 方式”。打开 MS-DOS 窗口后，便可以切换到程序所在的目录中，然后输入该程序的名称来运行它。图 G.12 是 Sample 程序在“MS-DOS 方式”窗口中的运行情况。



图 G.12　在“MS-DOS 方式”窗口中执行程序

G.5　总　结

阅读本附录后，您应该能够安装并使用 Dev-C++编译器。Dev-C++还有很多本附录中没有介绍的特性，有关如何使用该 IDE 的更详细的信息，请参阅它的帮助文档和其他文件。



注意：Dev-C++的帮助文档中有一个 FAQ（常见问题）列表，其中包含您可能遇到的很多问题及其答案。